



PROCEDURAL DESTRUCTION

Dennis Gustafsson
Mediocre

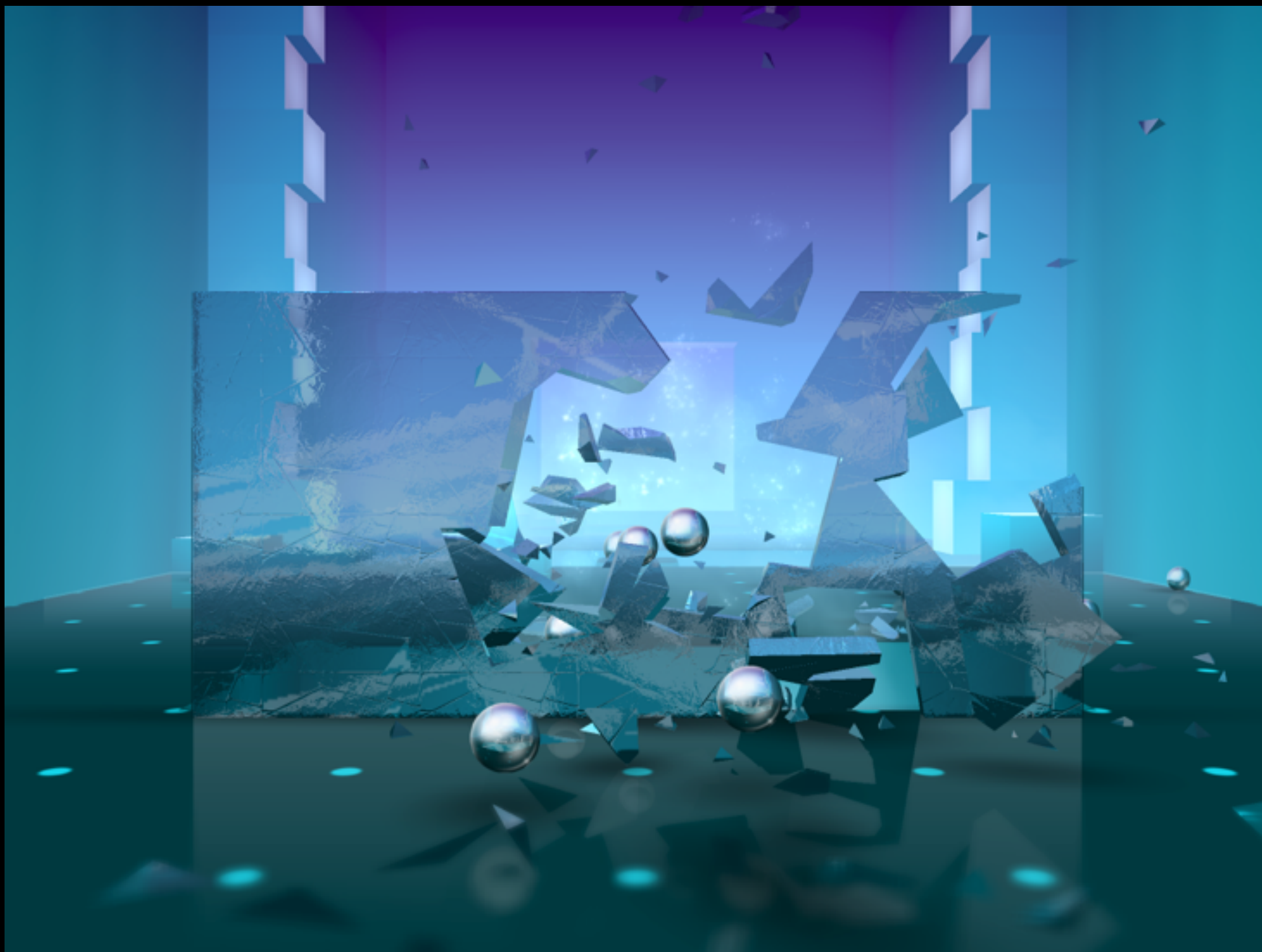
SMASH

HIT



SMASH

HIT



SMASH

HIT

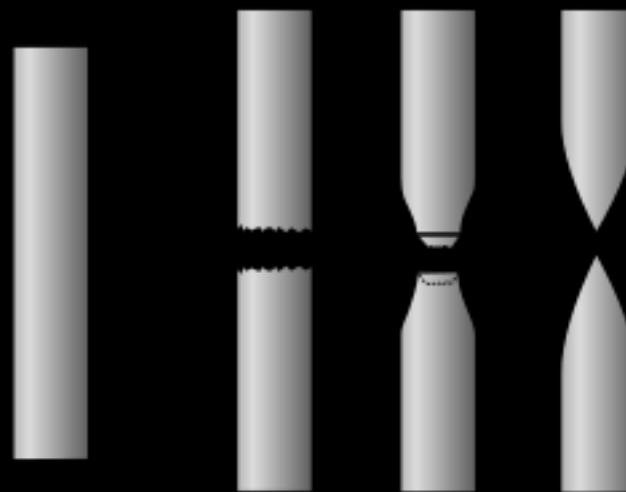




“A fracture is the separation of an object or material into two or more pieces under the action of stress”

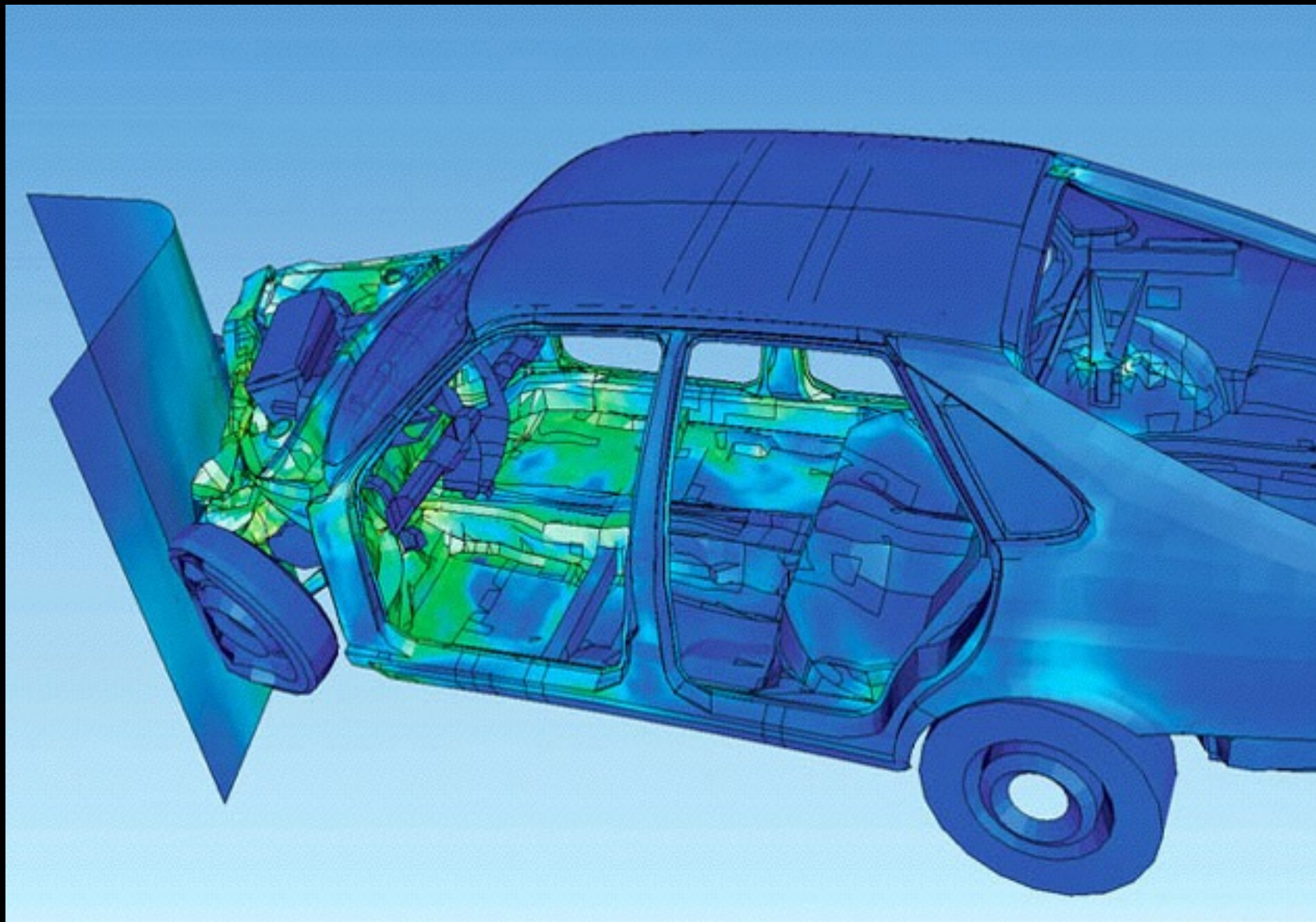


Brittle fracture vs ductile fracture





Finite Element Method



SMASH

HIT



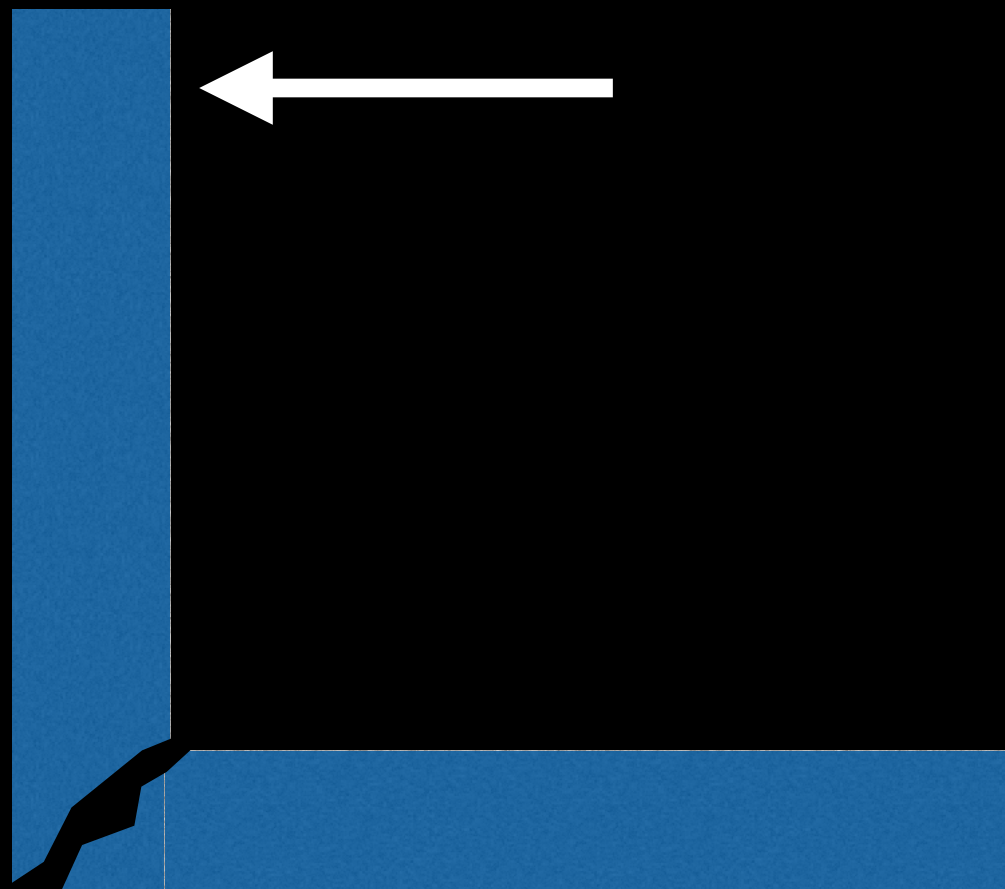
SMASH

HIT



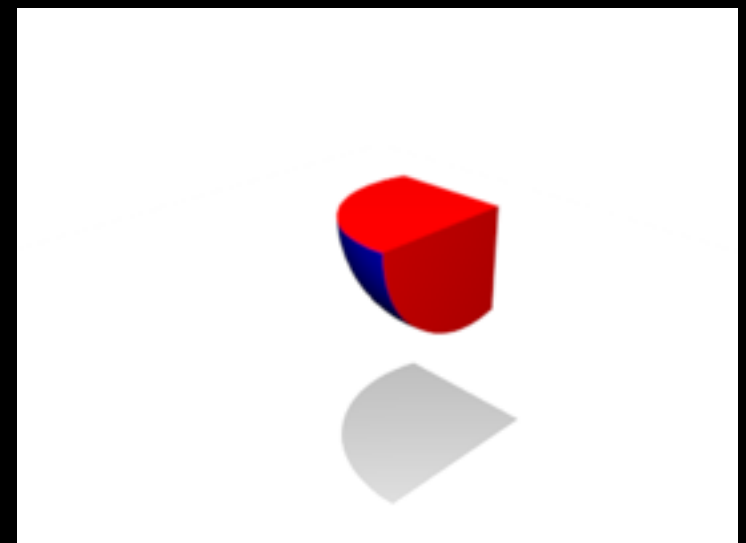
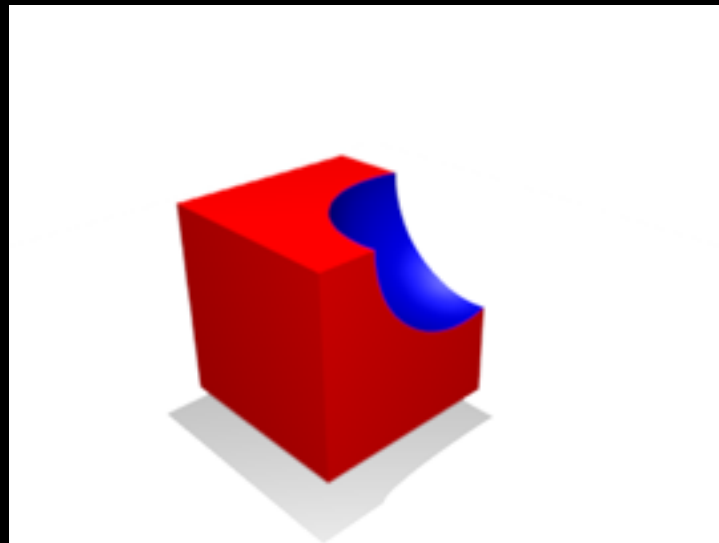
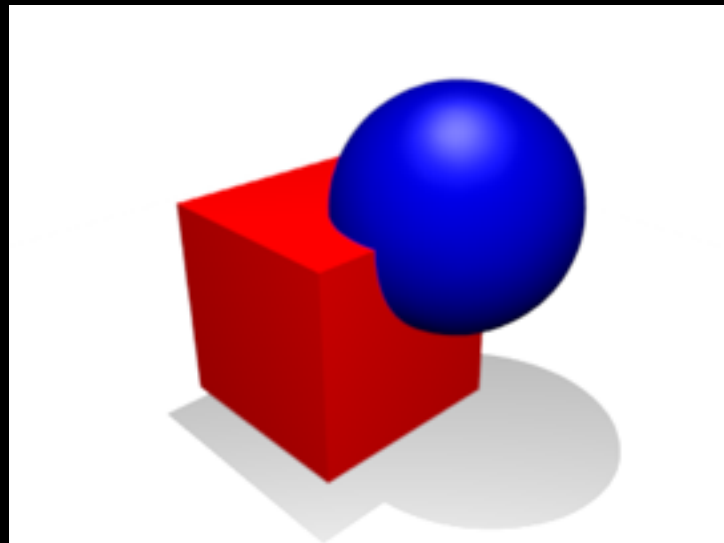
SMASH

HIT



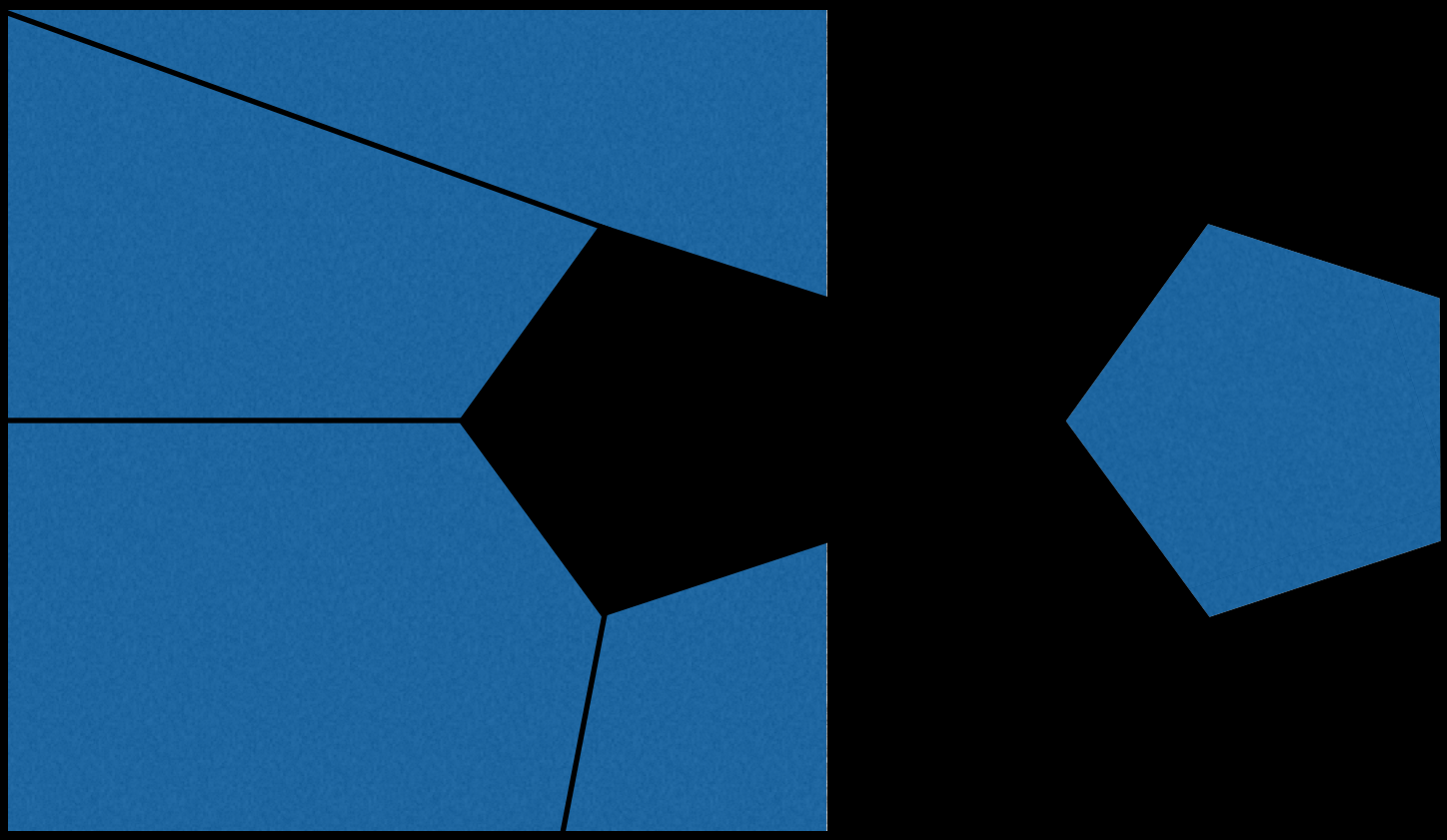


Boolean CSG operations



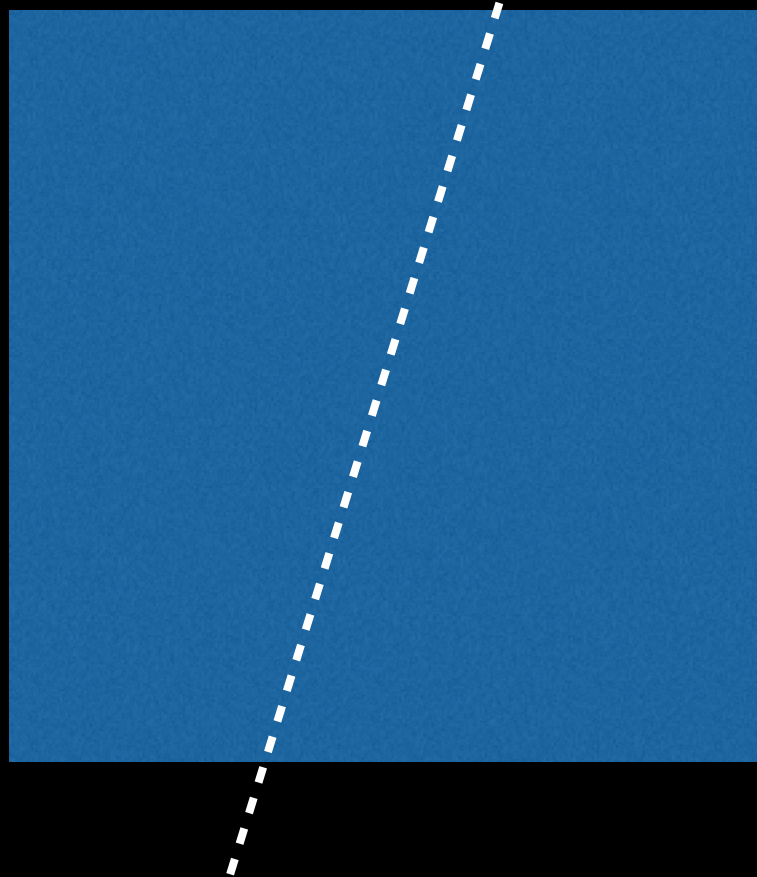
SMASH

HIT



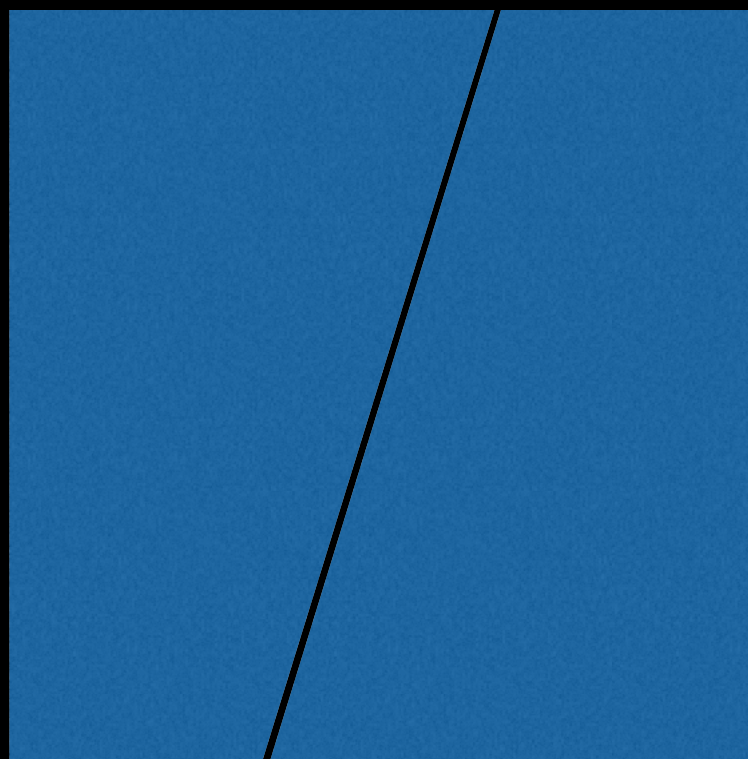
SMASH

HIT



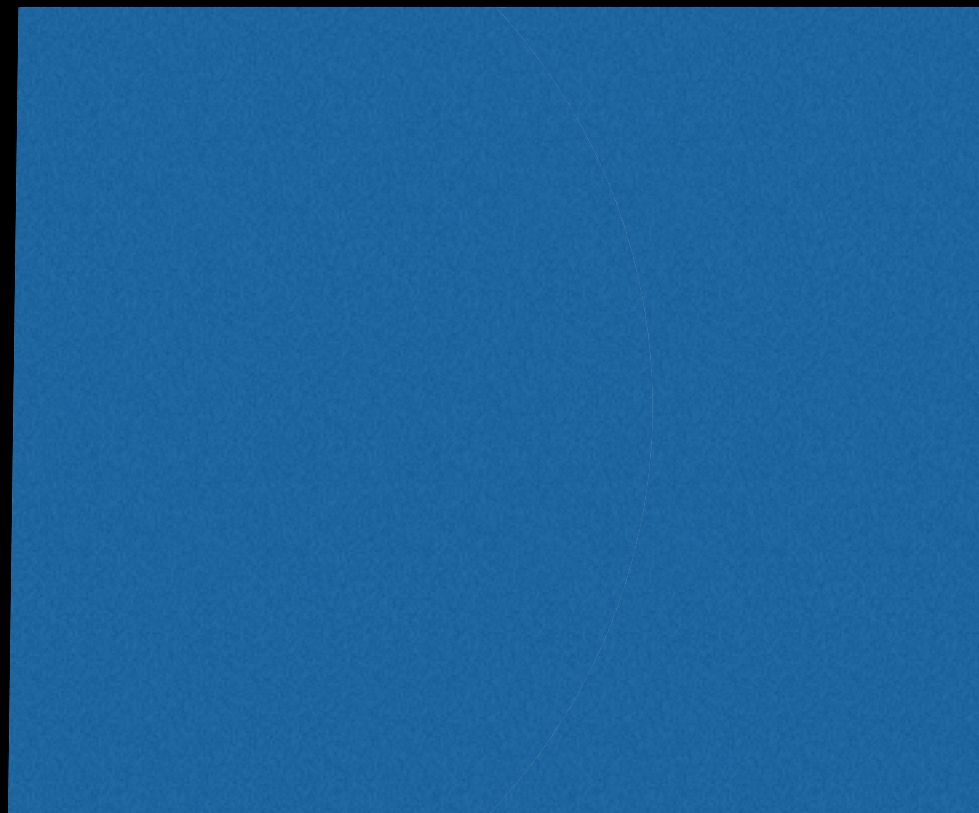
SMASH

HIT



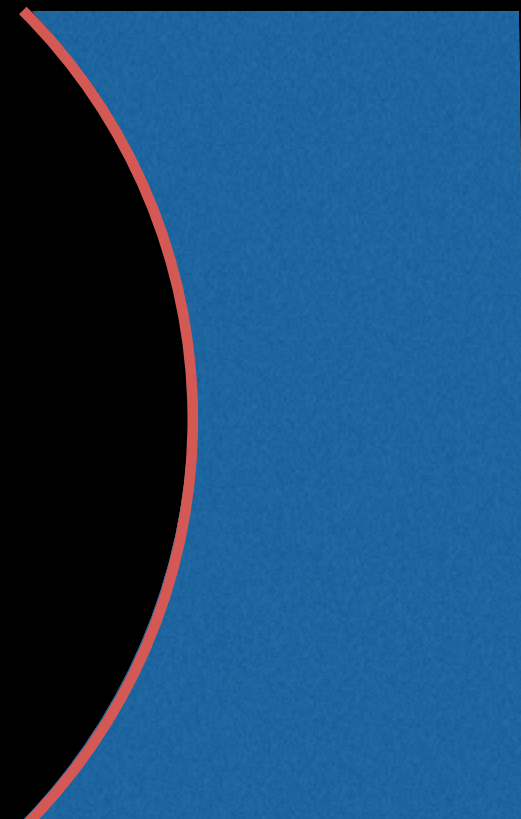
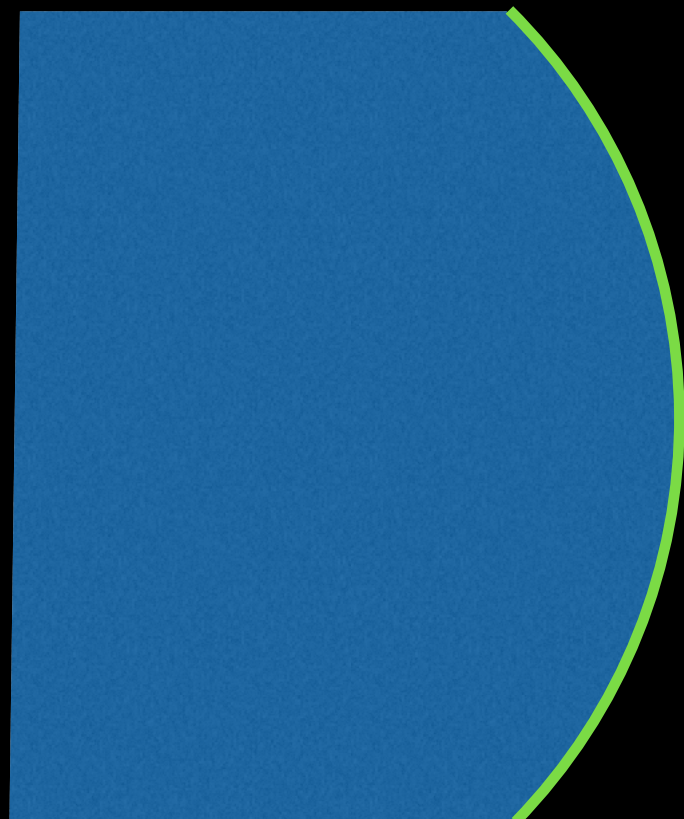
SMASH

HIT



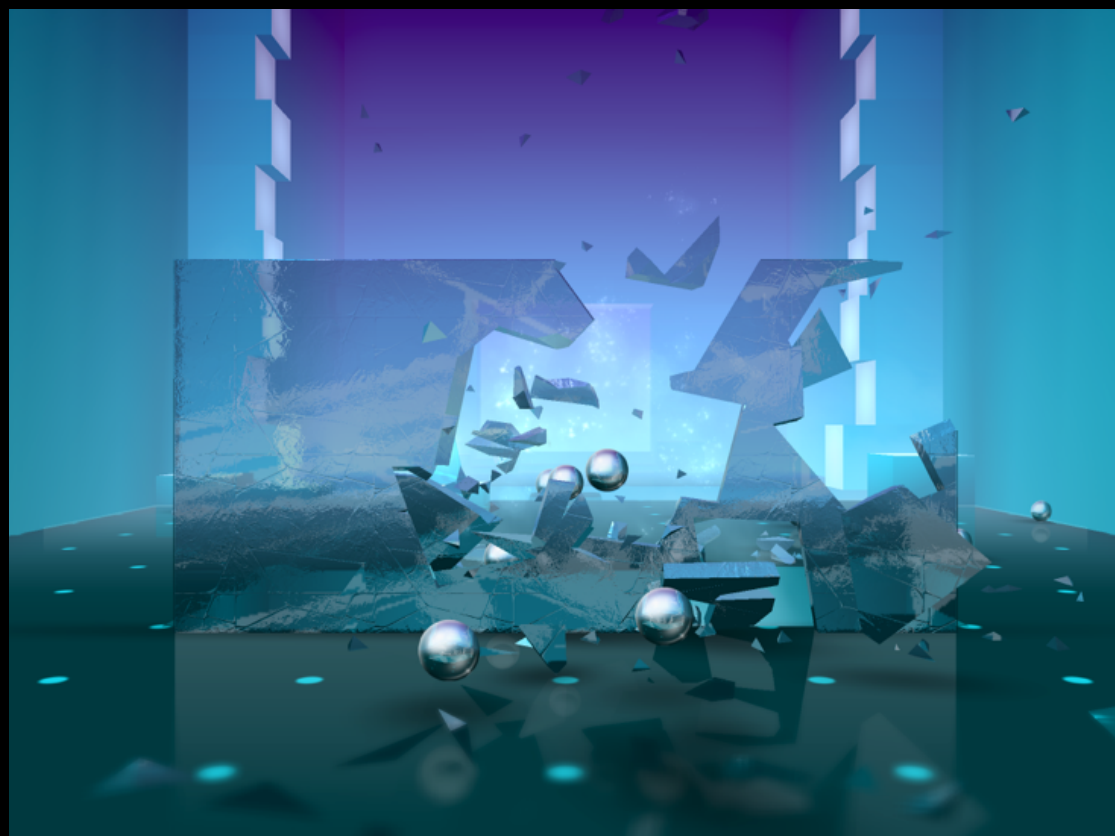
SMASH


HIT



SMASH

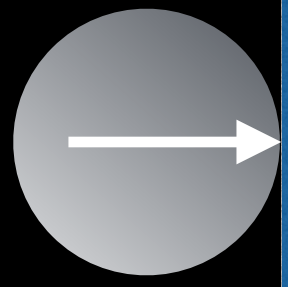
HIT



 Nathan nfm/ Wikipedia

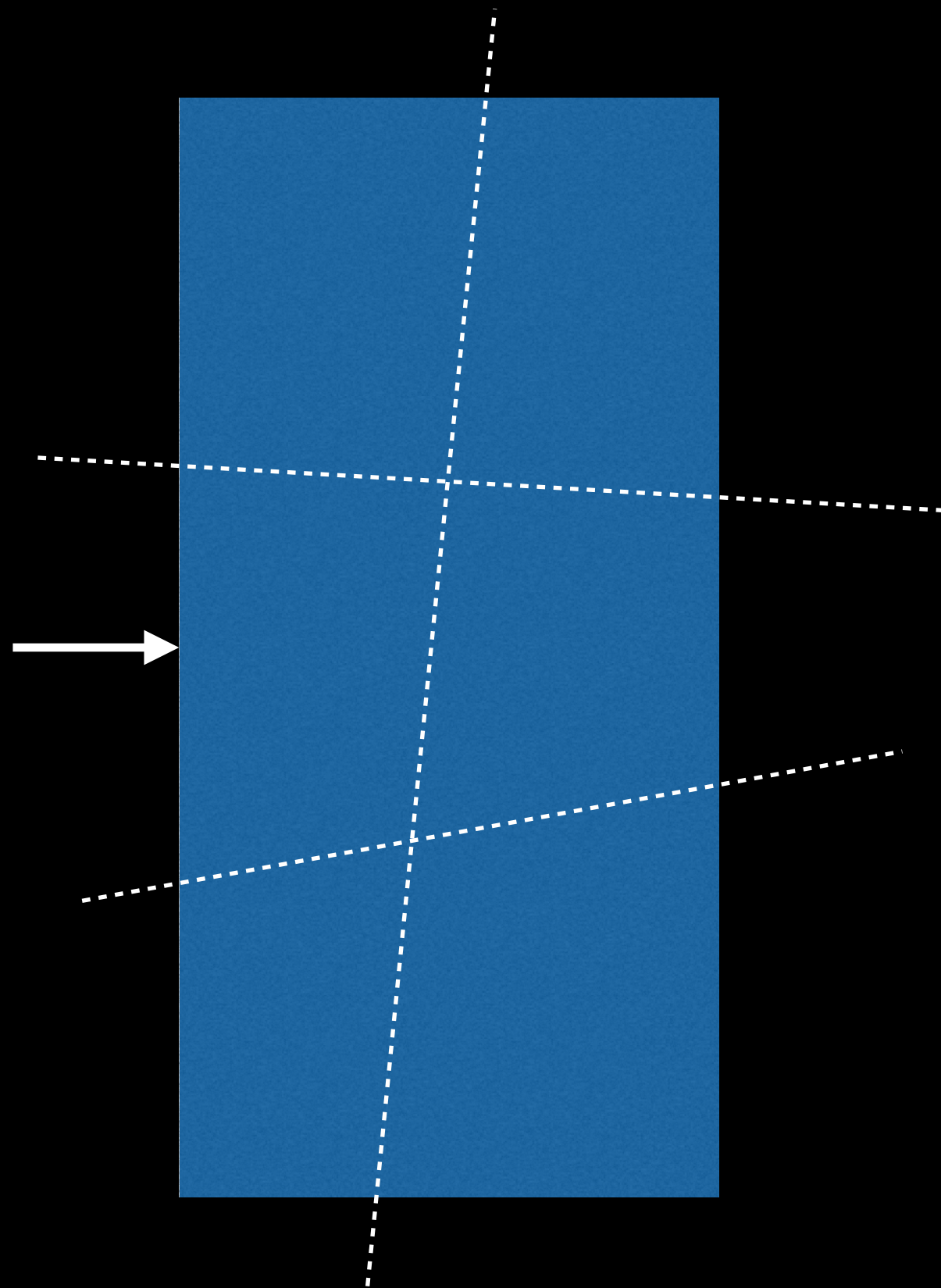
SMASH

HIT



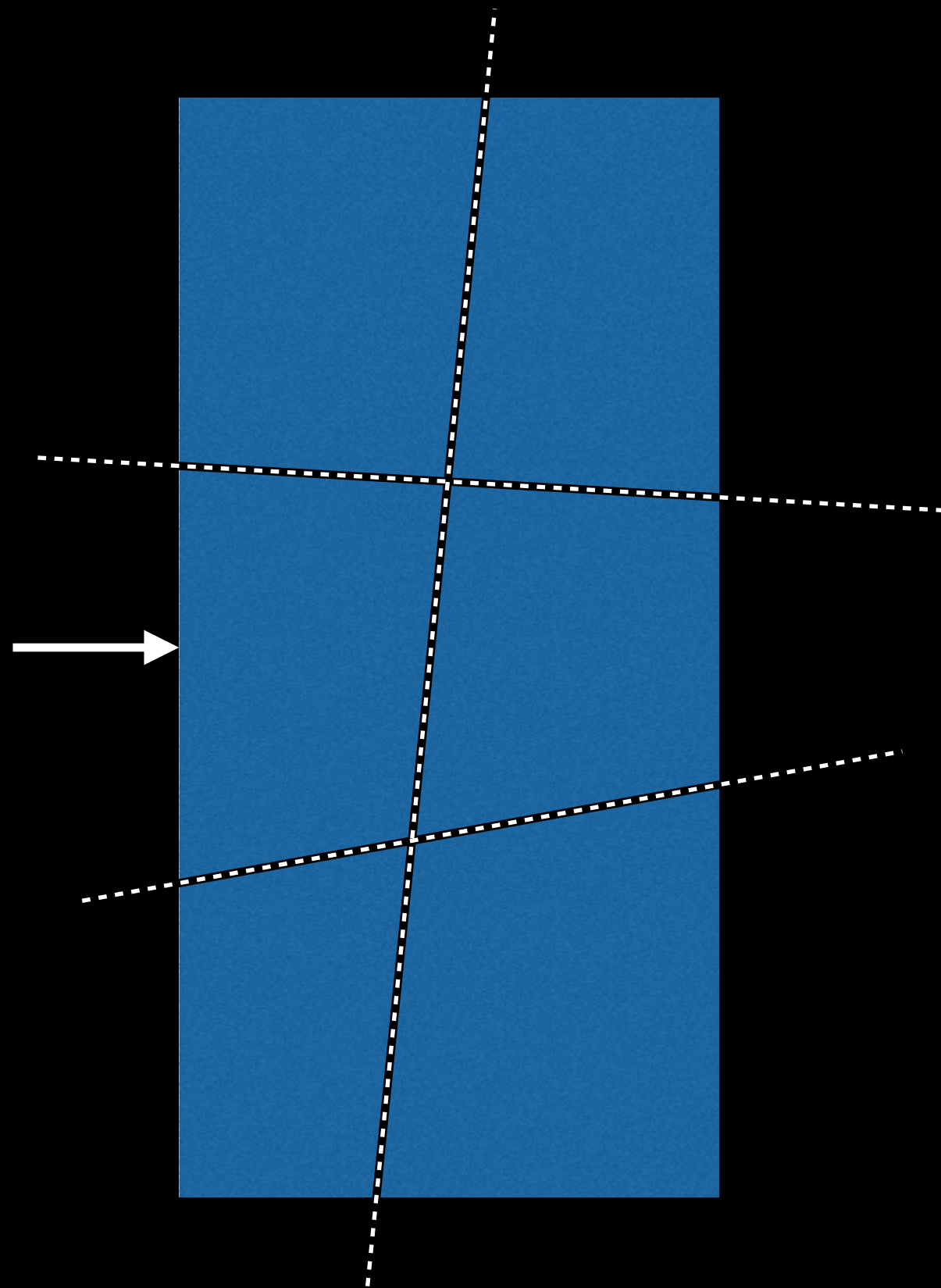
SMASH

HIT



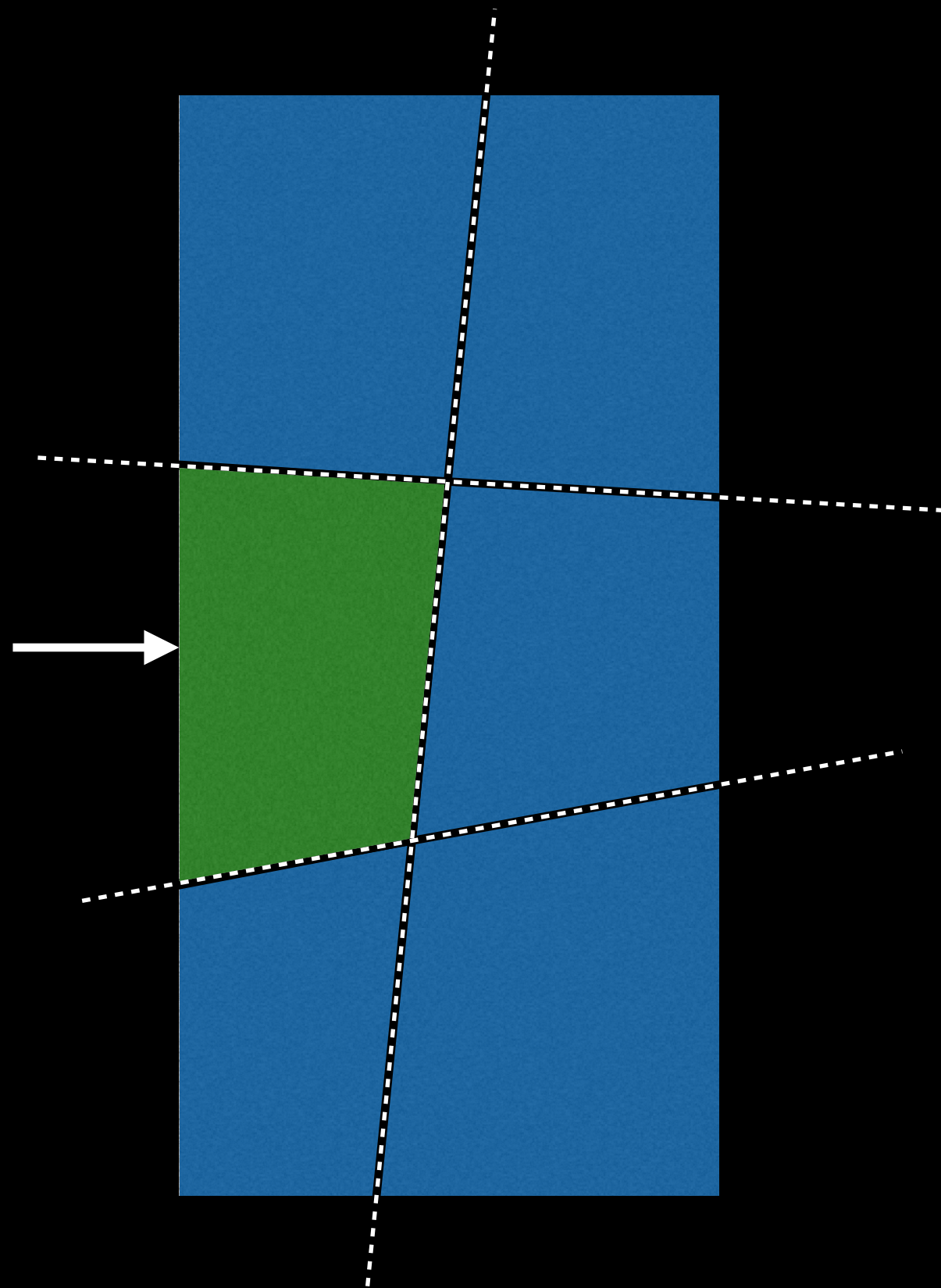
SMASH

HIT



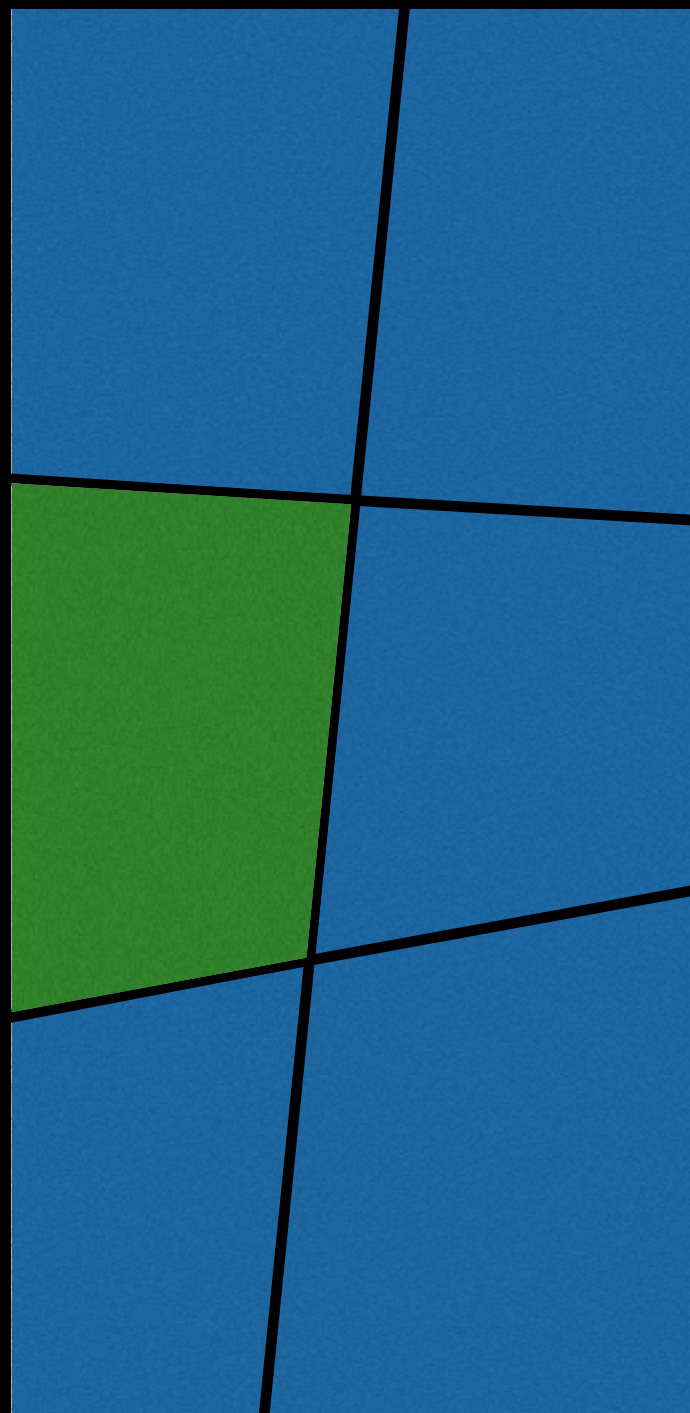
SMASH

HIT



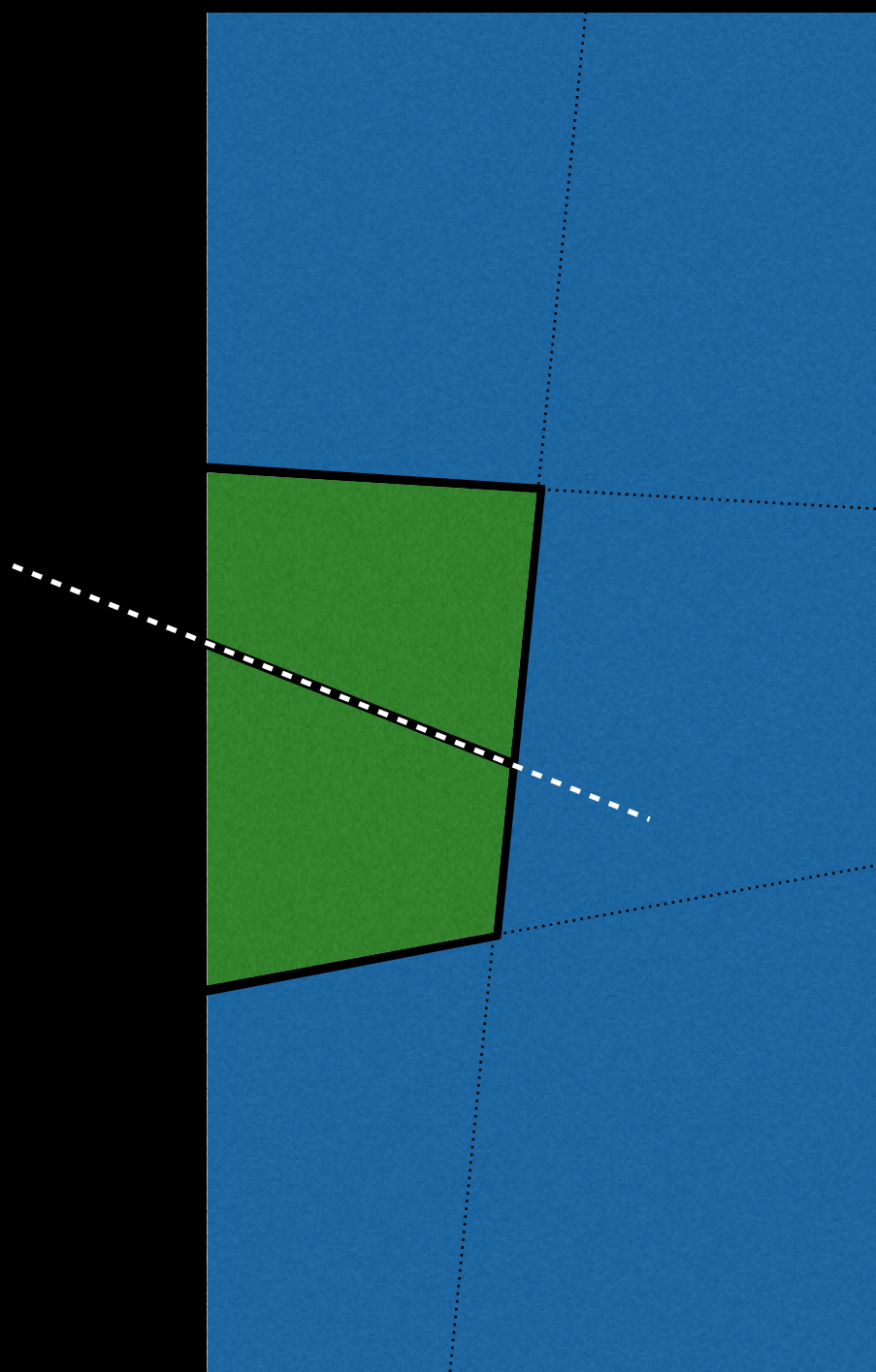
SMASH

HIT



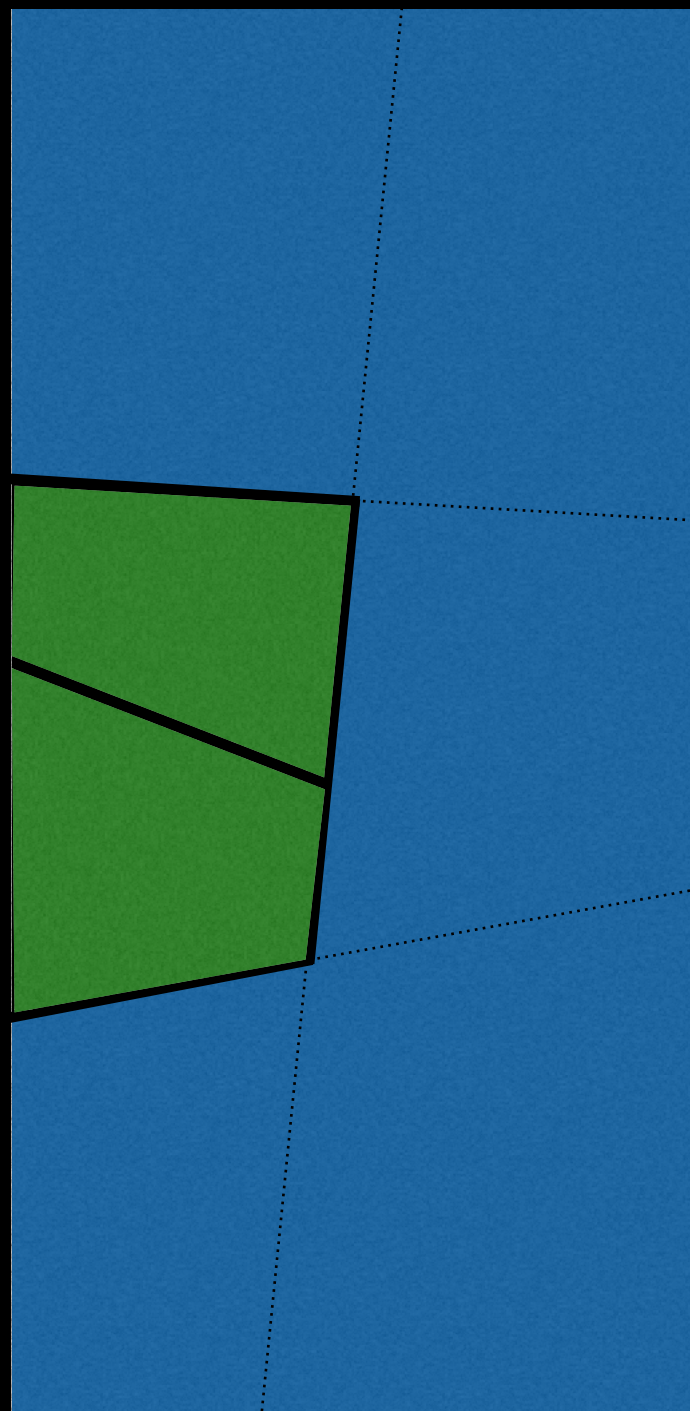
SMASH

HIT



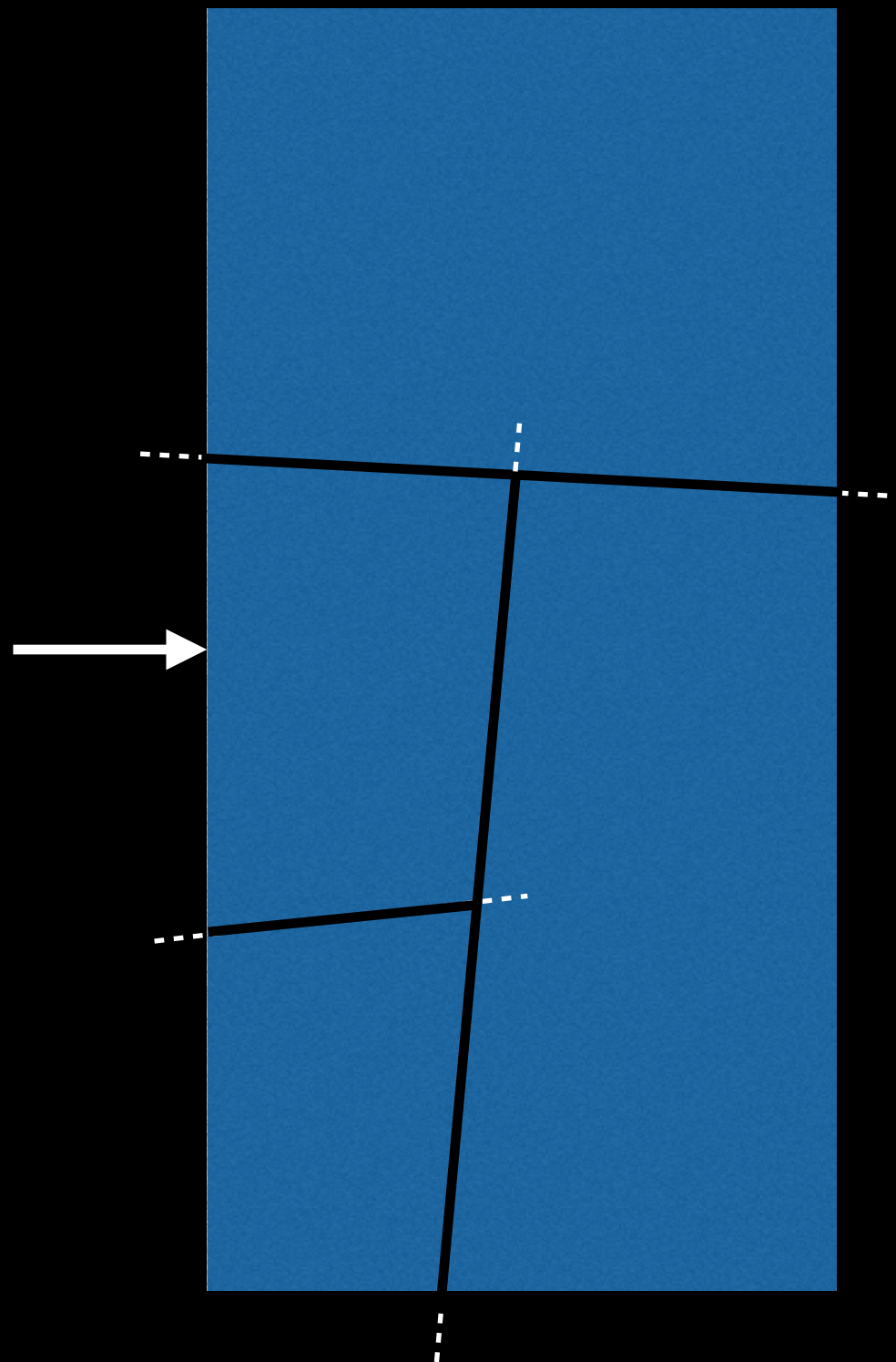
SMASH

HIT



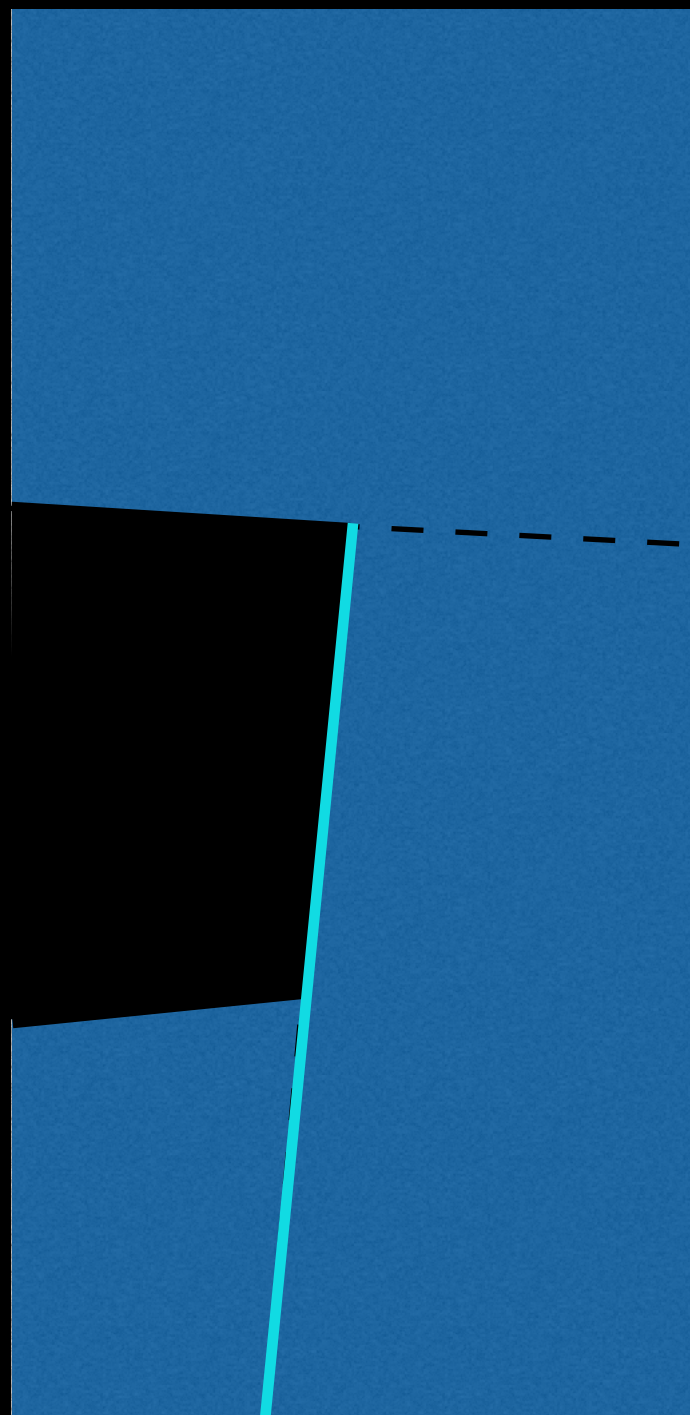
SMASH

HIT



SMASH

HIT



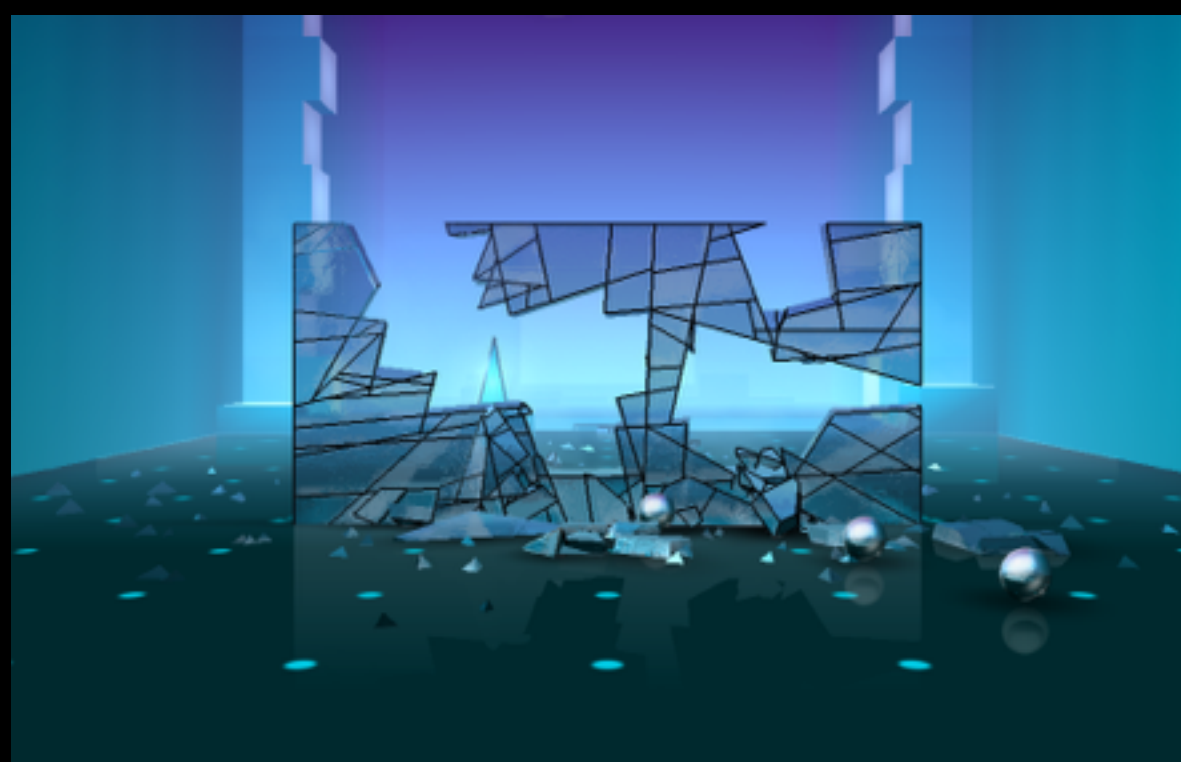
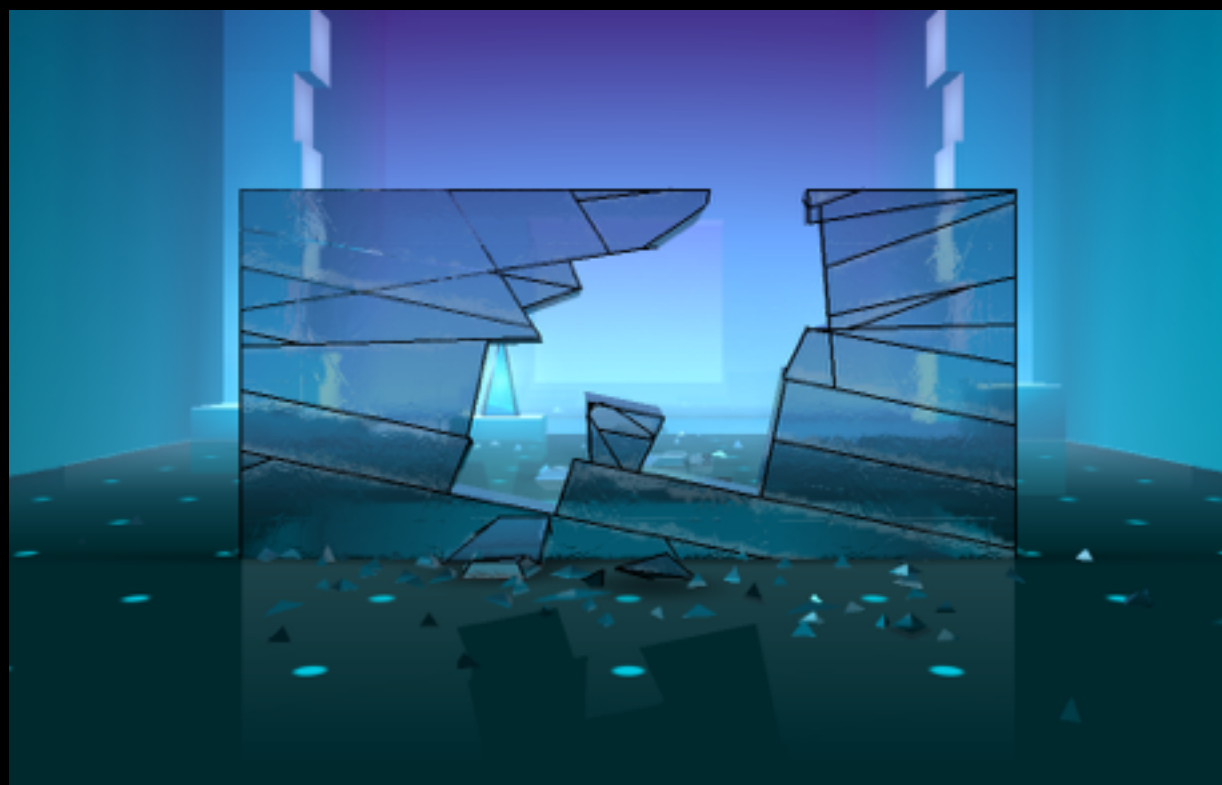
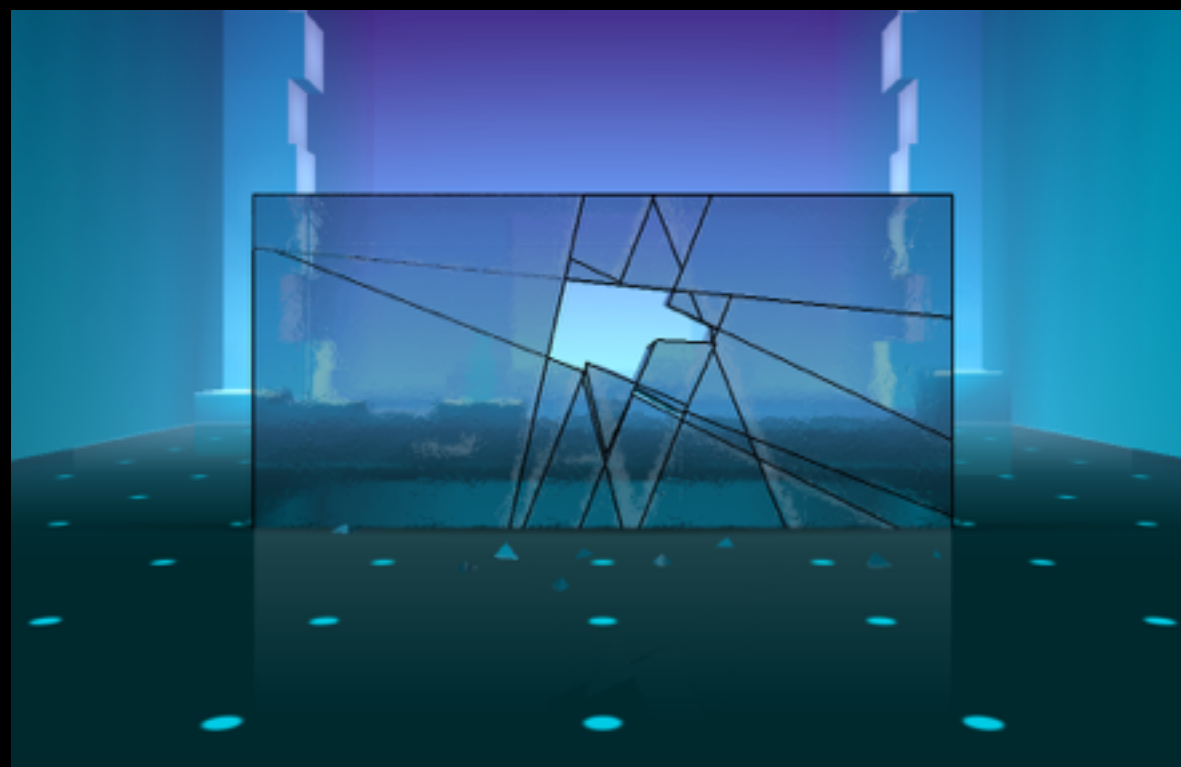
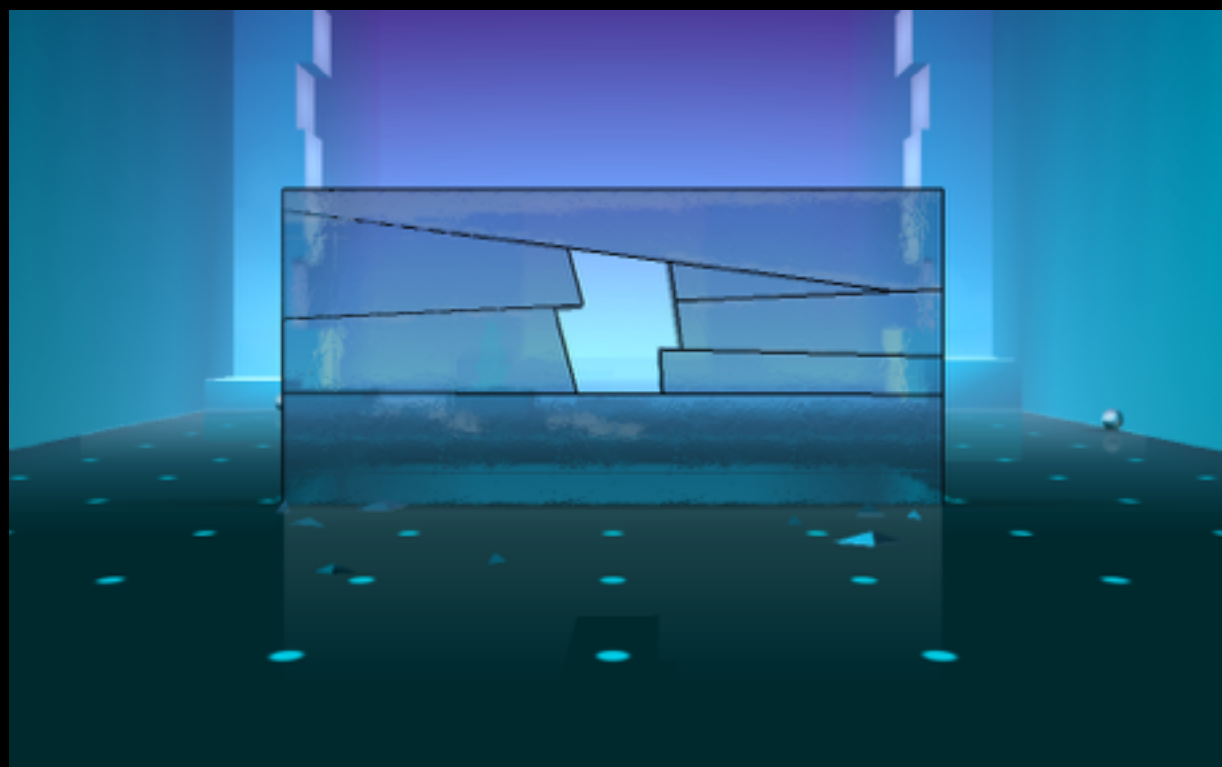
SMASH

HIT



SMASH

HIT

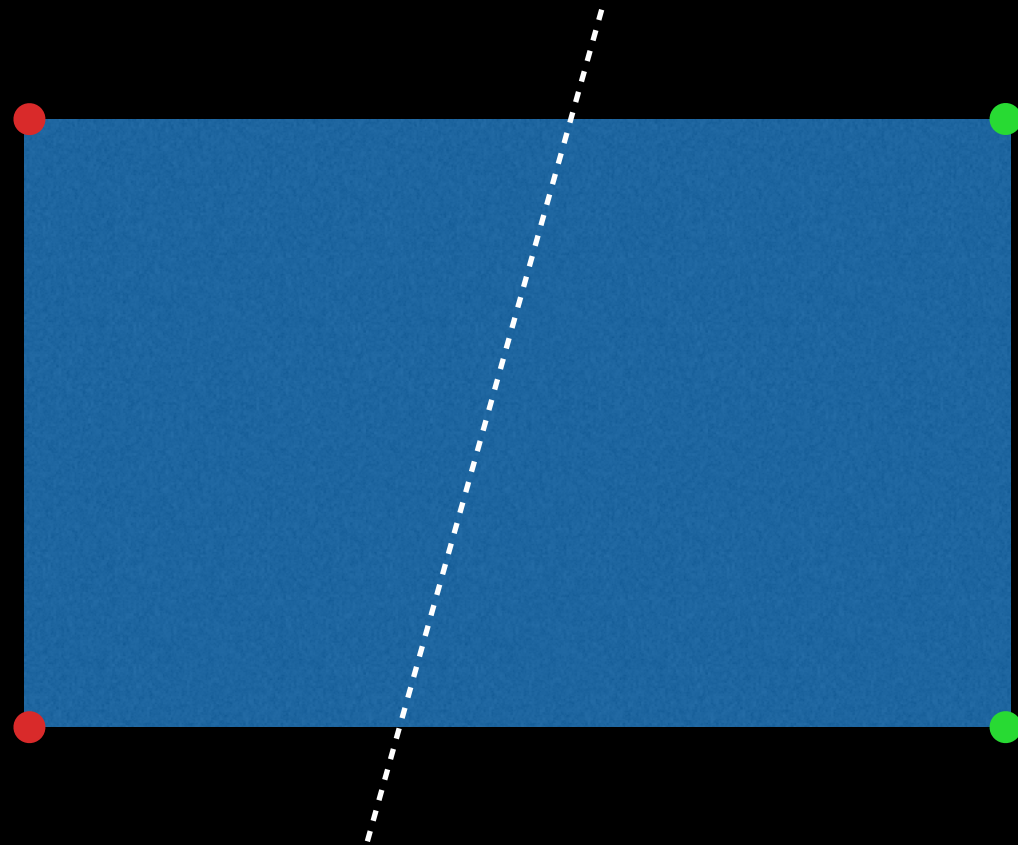




Robust convex plane splitting

SMASH

HIT





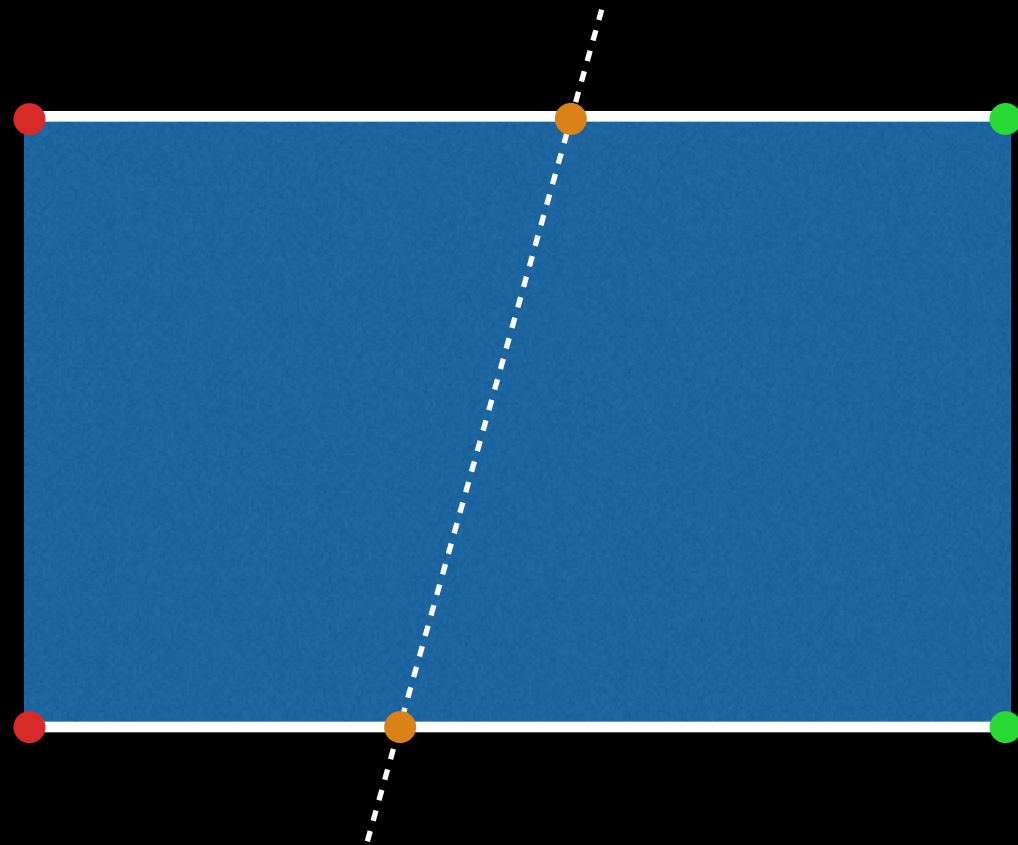
For each vertex v

 Determine if v is above or below the splitting plane

Next

SMASH

HIT

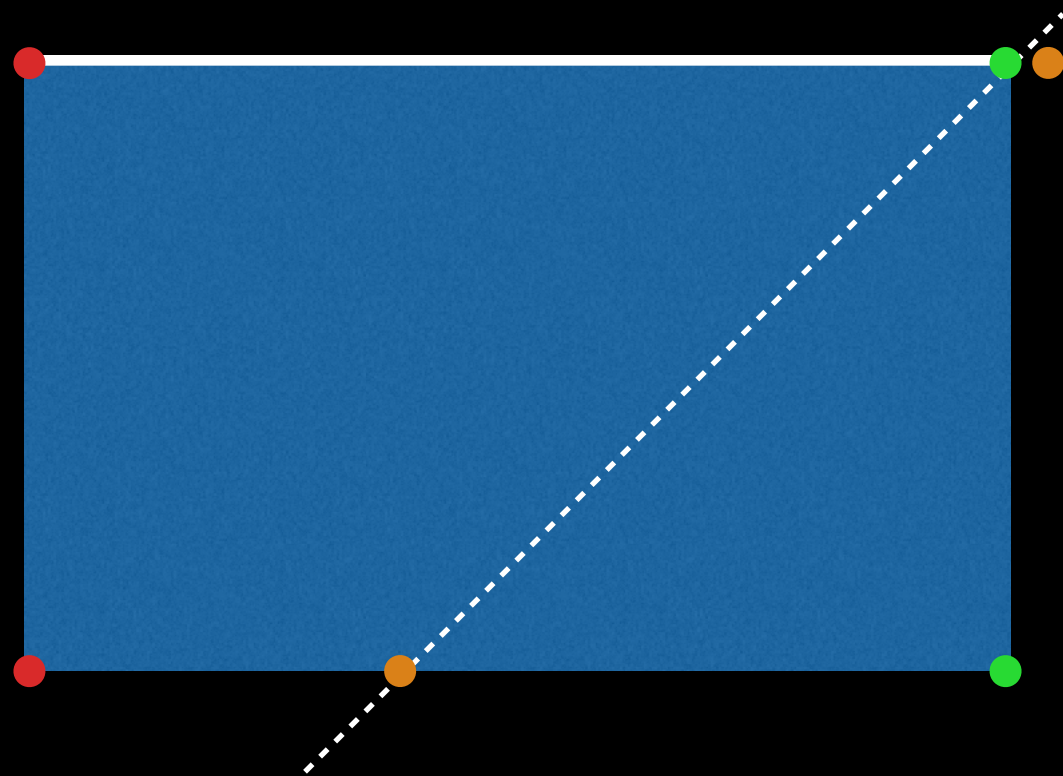




For each edge e that has one vertex above and one below
Find intersection point with plane
Next

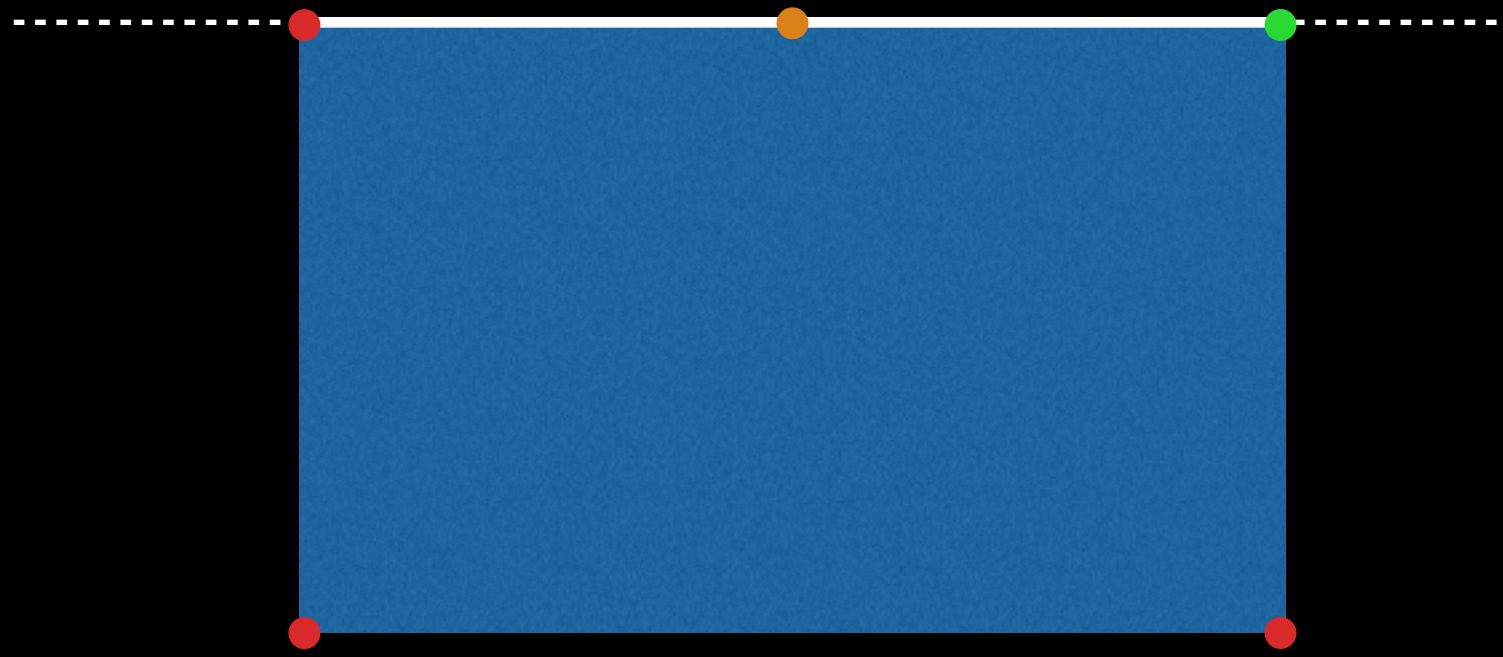
SMASH

HIT





?





For each edge e that has one vertex above and one below
Find intersection point p with plane

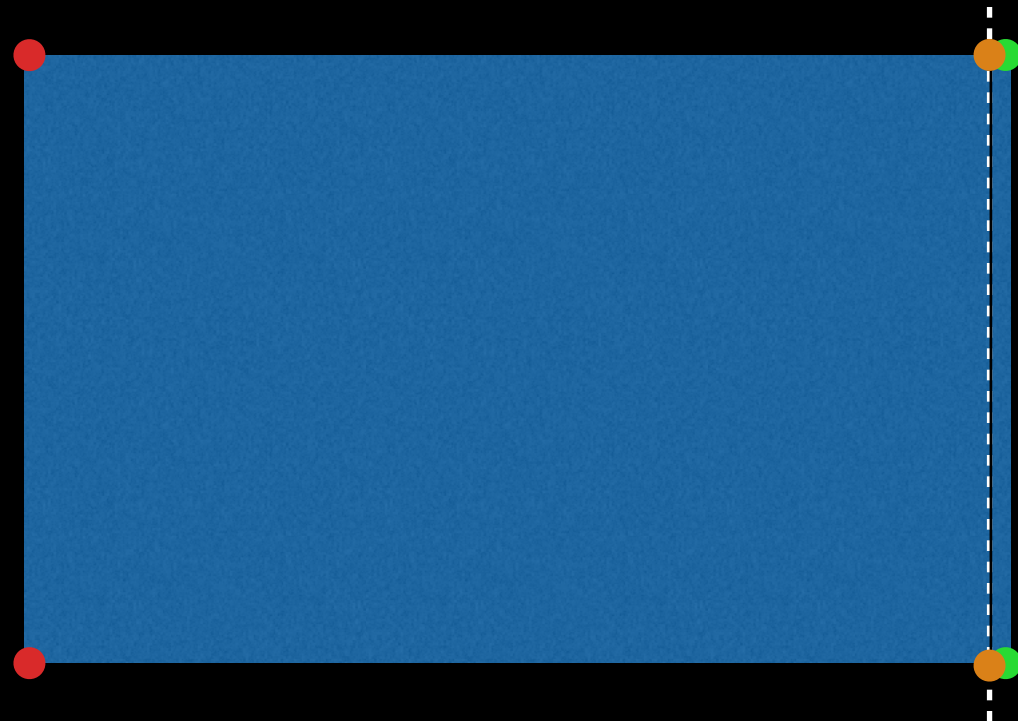
Project p onto edge

Clamp p to lie in between the two vertices

Next

SMASH

HIT

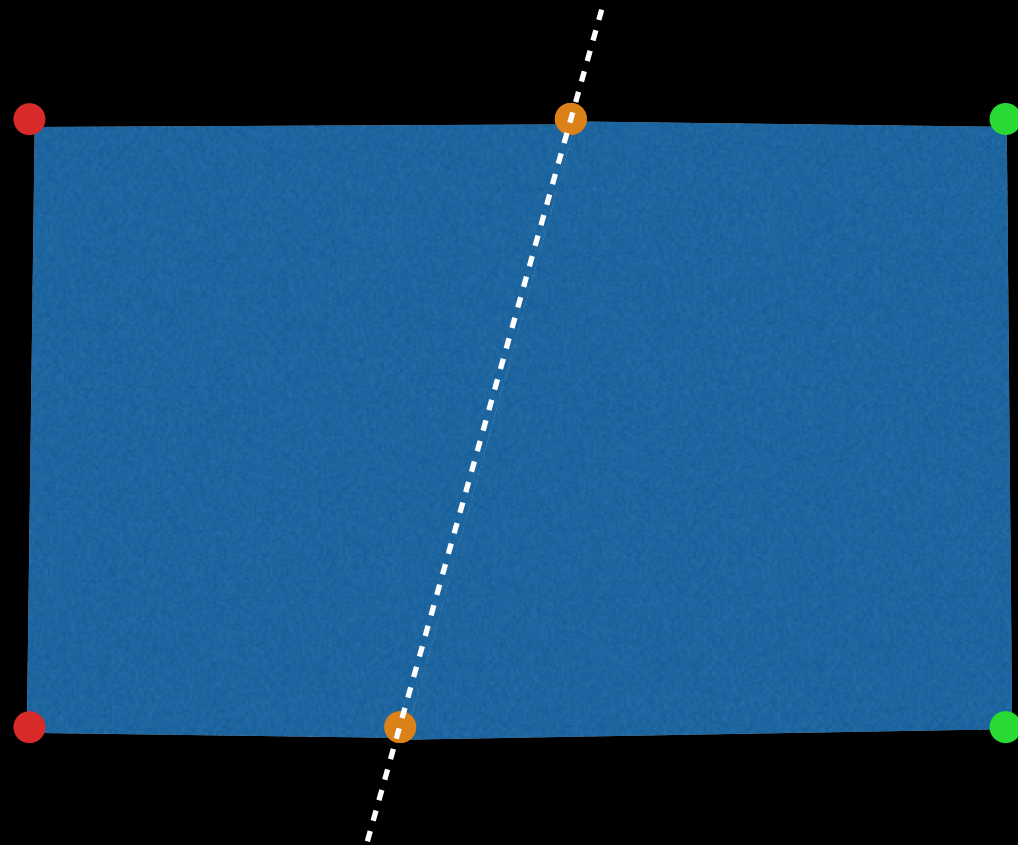




For each edge e that has one vertex above and one below
Find intersection point with plane
Project p onto edge
Clamp p to lie in between the two vertices
If p is close to vertex, move it away from vertex
Next

SMASH

HIT

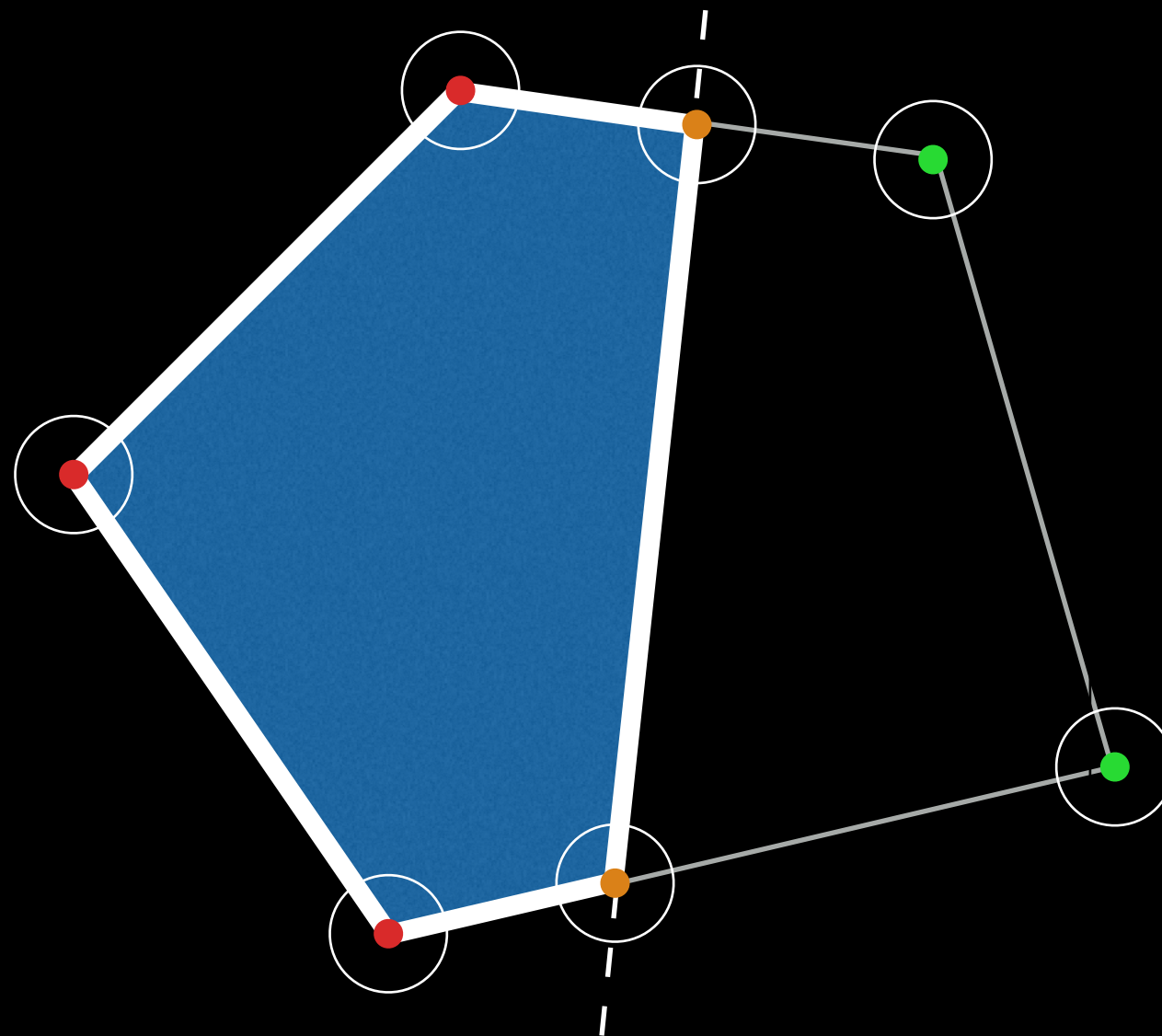




no math beyond this point

SMASH

HIT

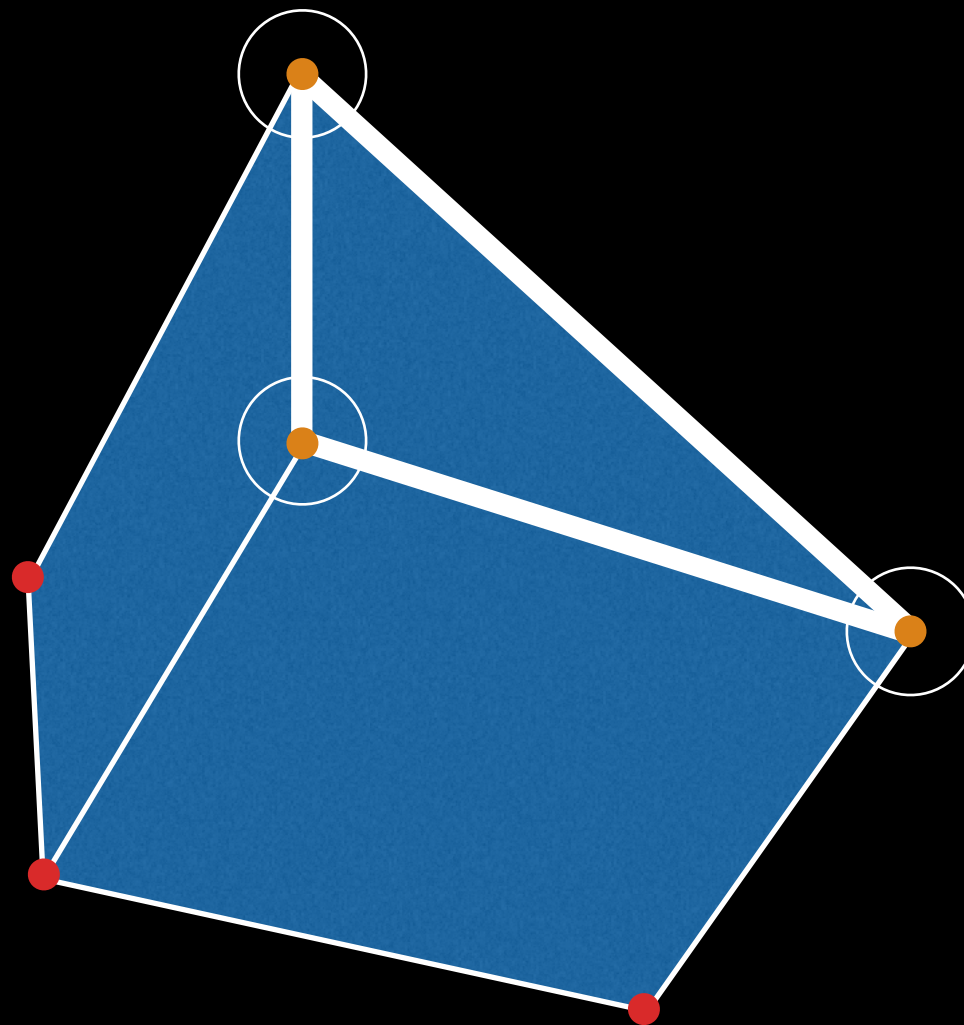




```
pick any vertex vStart on side A
do
    v = nextVertex
    if v is split vertex then
        do
            v = nextVertex
            until v is split vertex
        add new edge
    until v == vStart
```

SMASH

HIT





```
pick any split edge eStart in polyhedron A
do
    find connected split edge e
    reverse edge e and add to A
until e == eStart
```



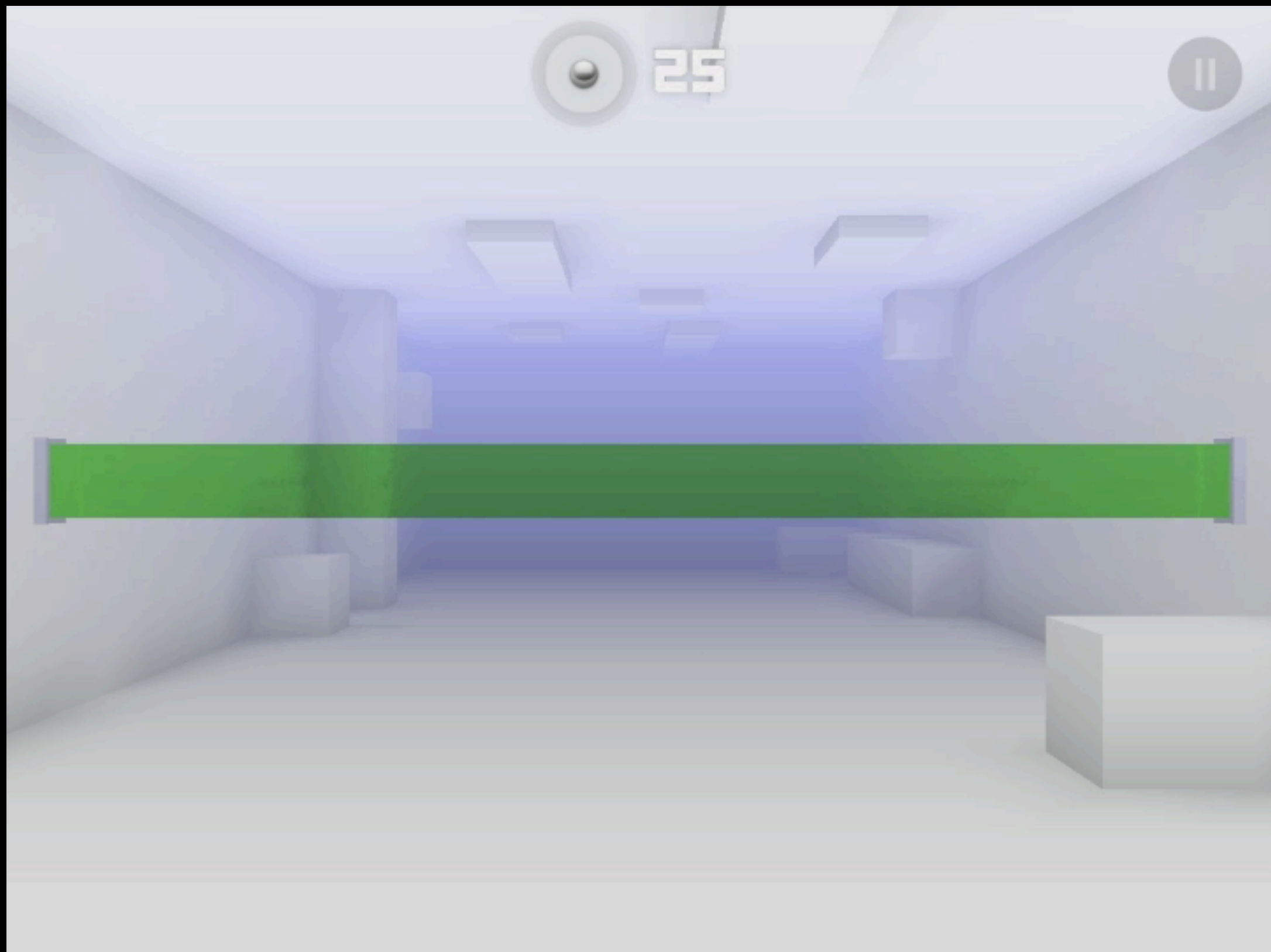
```
struct Vertex
{
    vec3 mPoint;
    ...
};

struct Face
{
    short int mEdge;
    ...
};

struct Edge
{
    short int mFace;
    short int mVertex;
    short int mNextEdge
    short int mOppositeEdge;
    ...
};
```

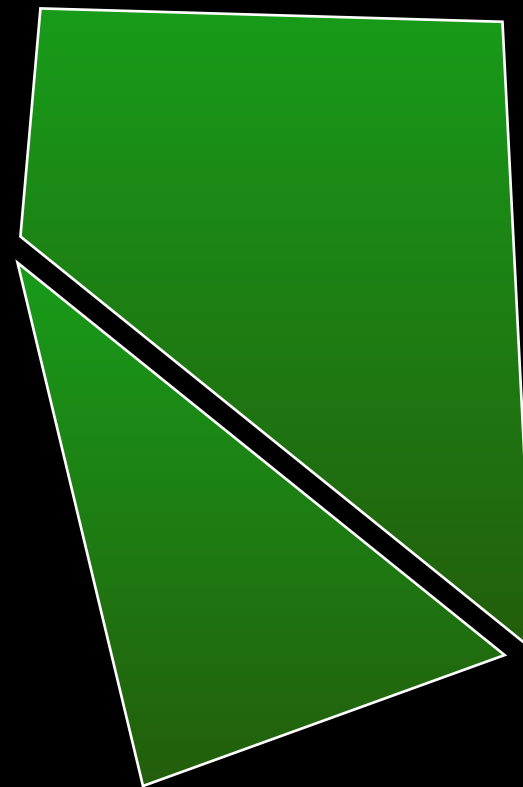
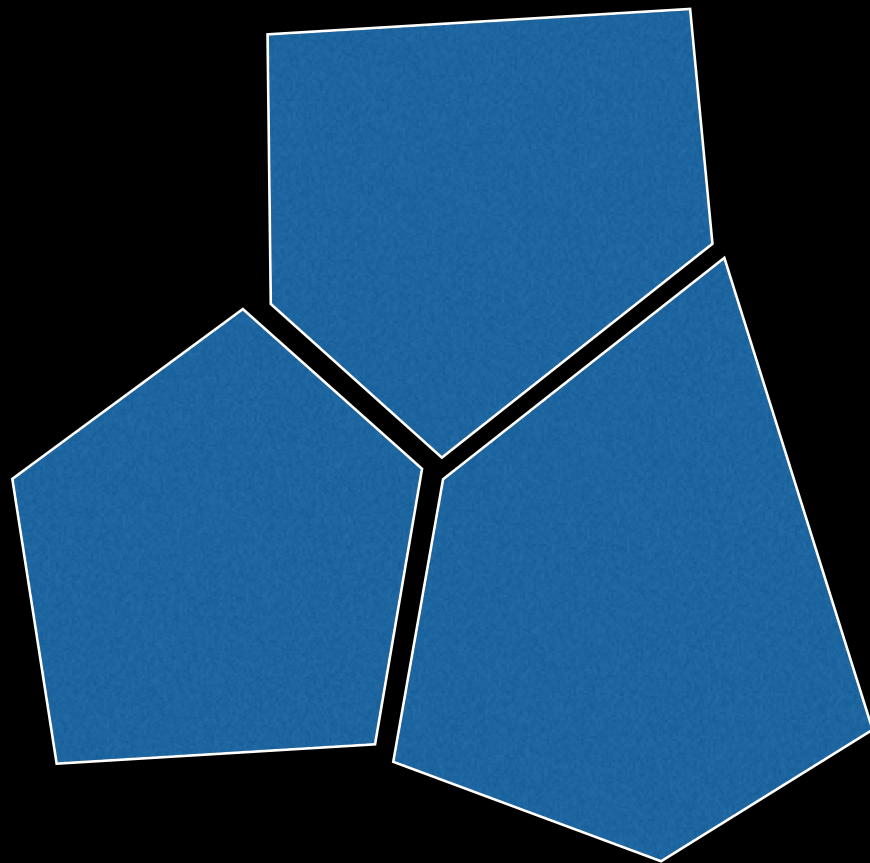
SMASH

HIT



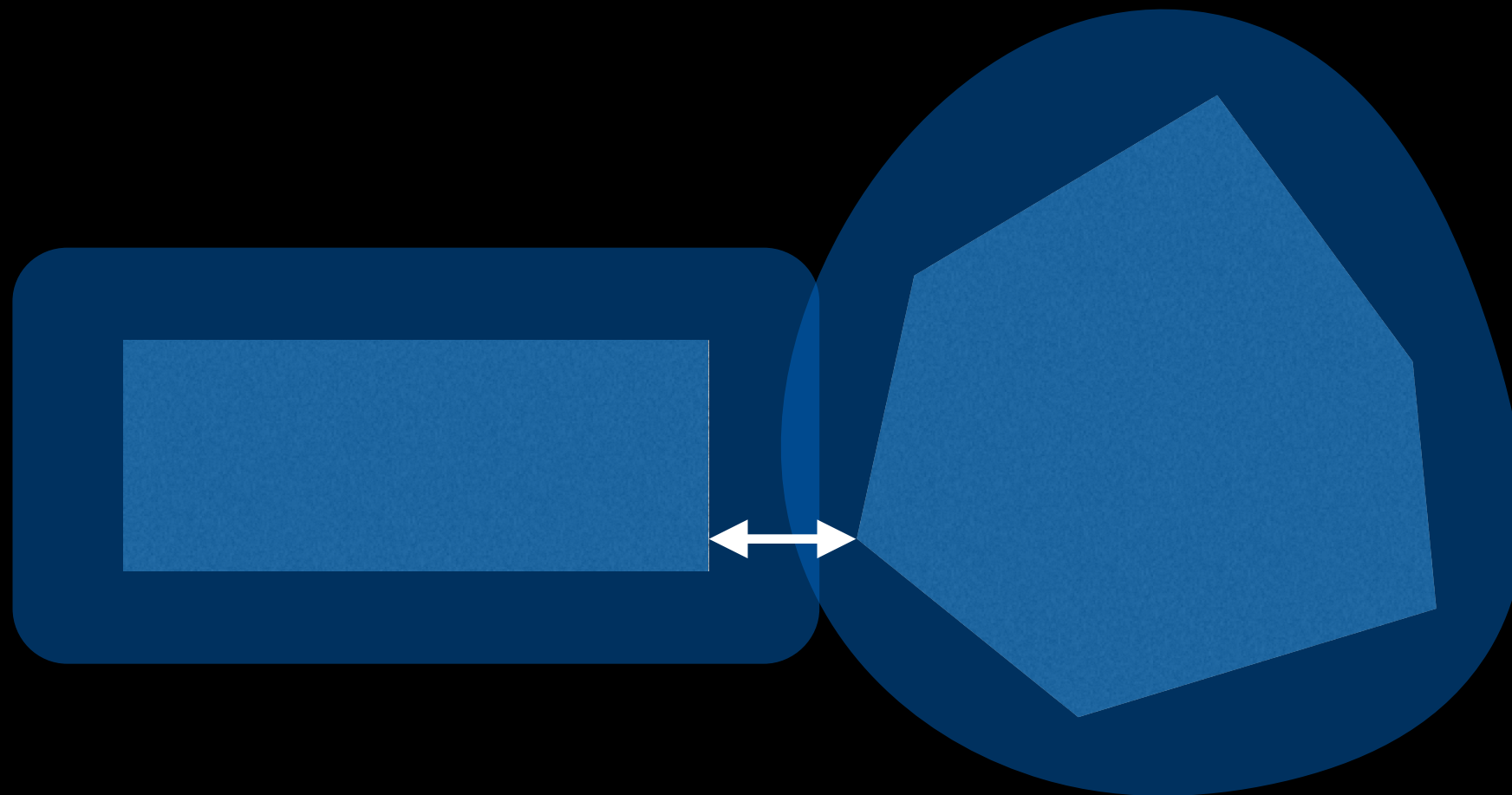
SMASH

HIT



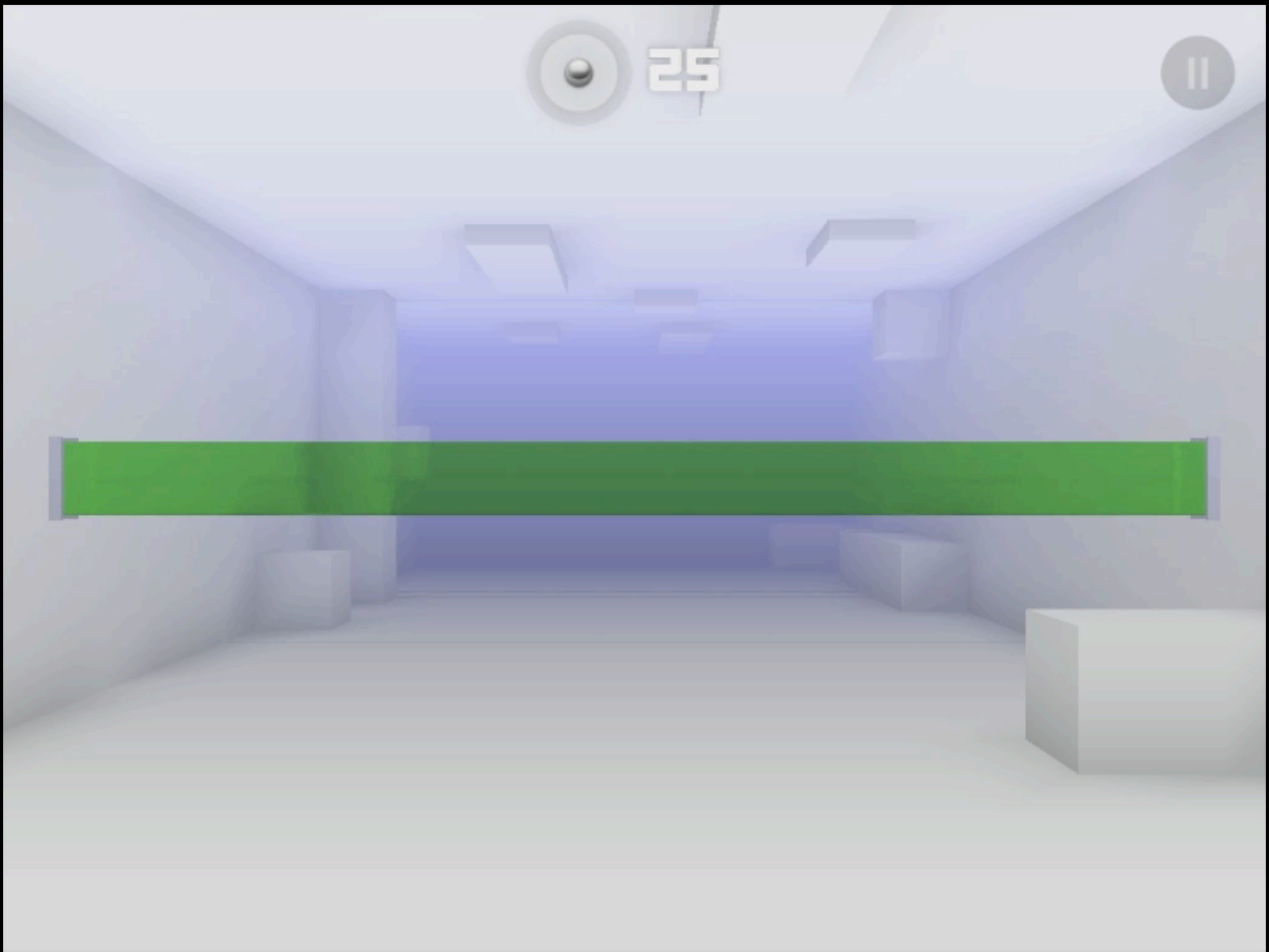


distance test
configuration space overlap test



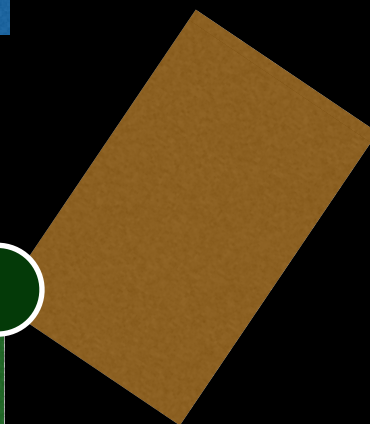
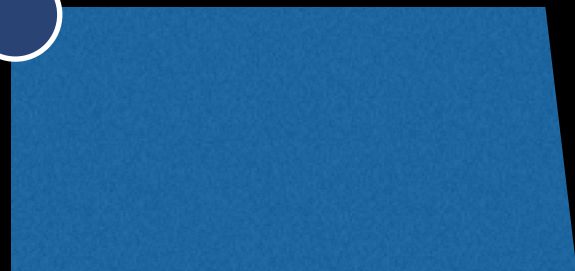
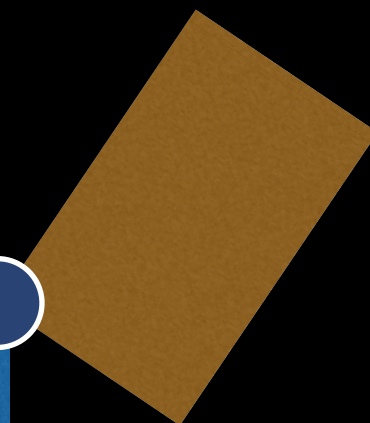
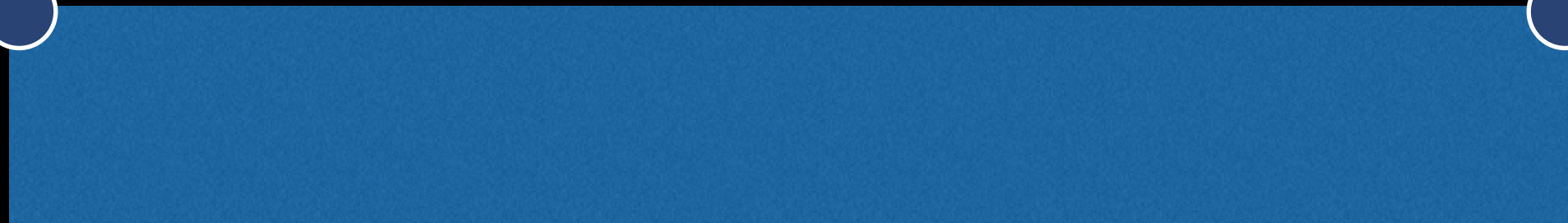
SMASH

HIT



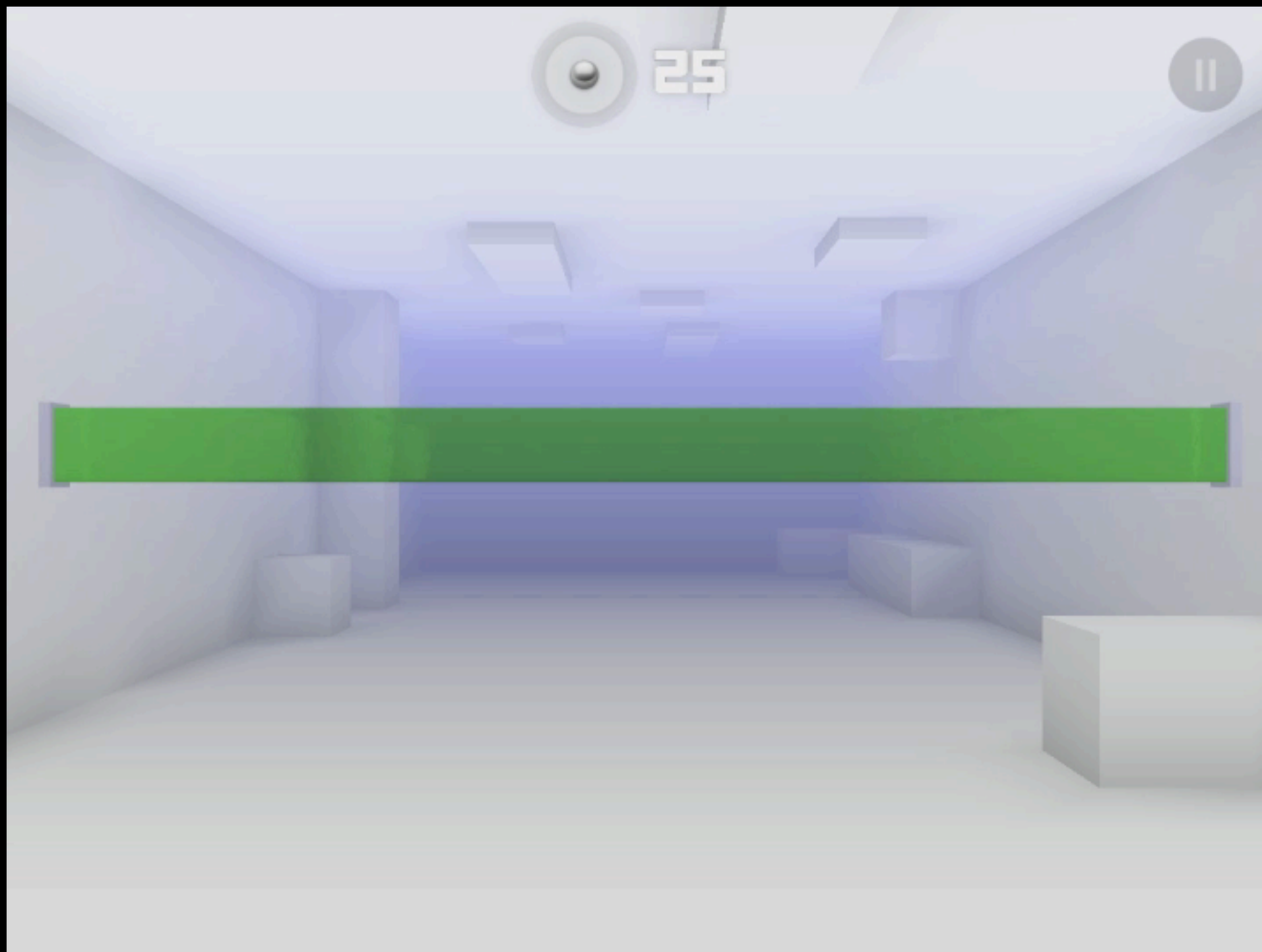
SMASH

HIT



SMASH

HIT

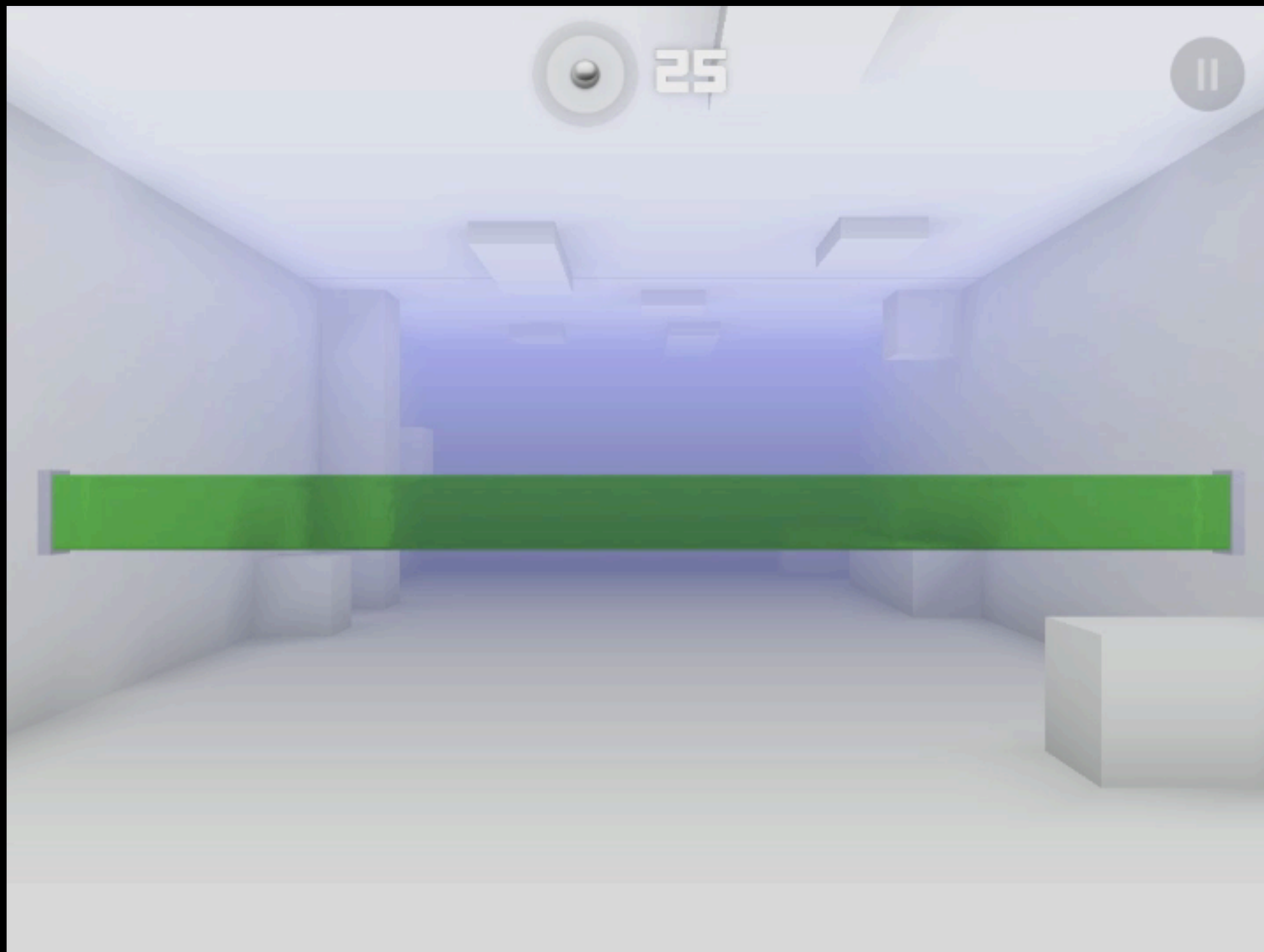




```
body->animate(...)
```

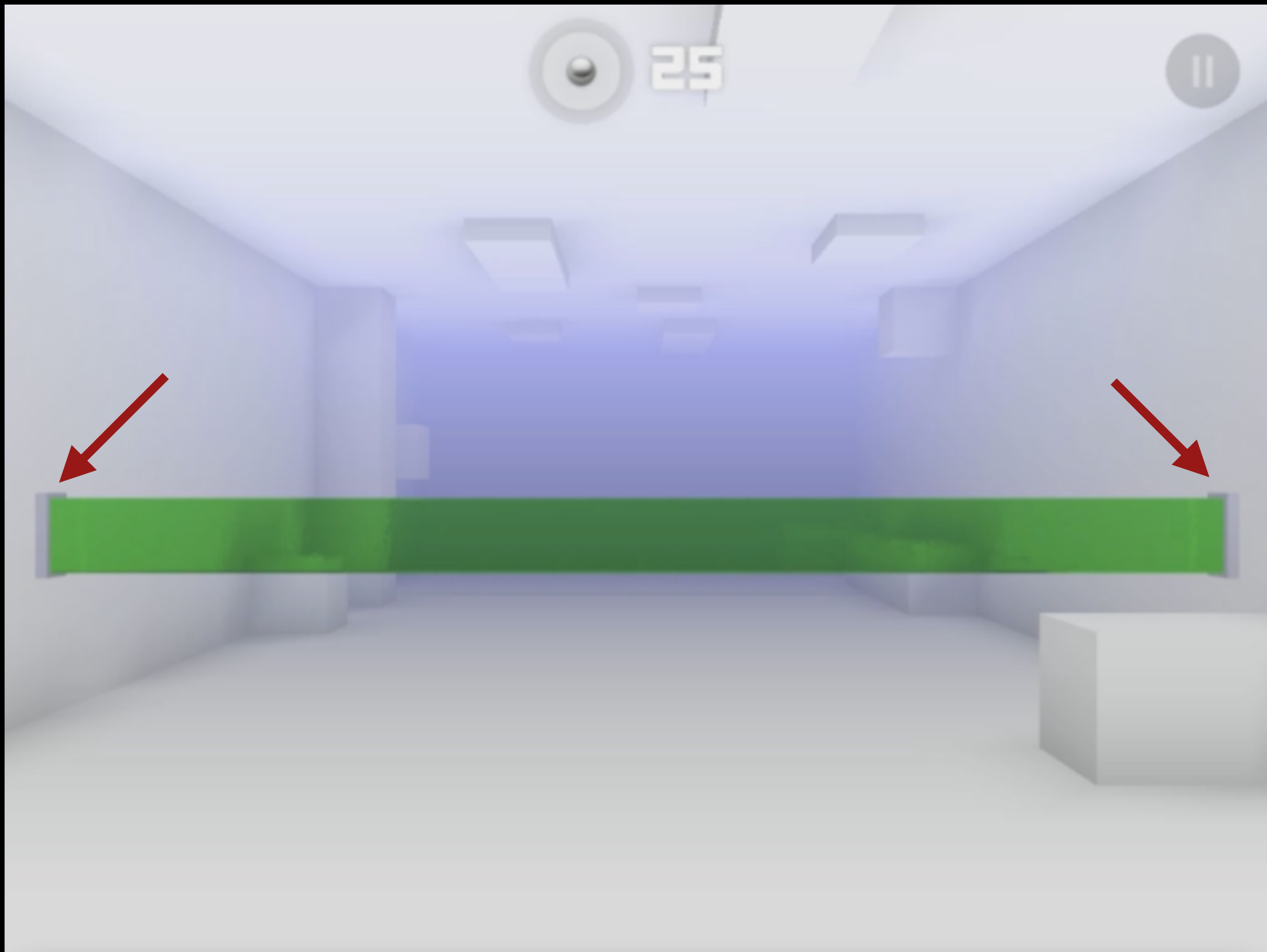

SMASH

HIT



SMASH

HIT





~~body->animate(...)~~

```
leftBody = leftShape->getBody()  
rightBody = rightShape->getBody()  
leftBody->animate(...)  
if rightBody != leftBody then  
    rightBody->animate(...)  
end
```



Shape-centric physics engine



```
physicsStep()  
  
for each contact c  
    if (c.impulse > limit)  
        fracture(body)  
    end  
end
```

SMASH

HIT

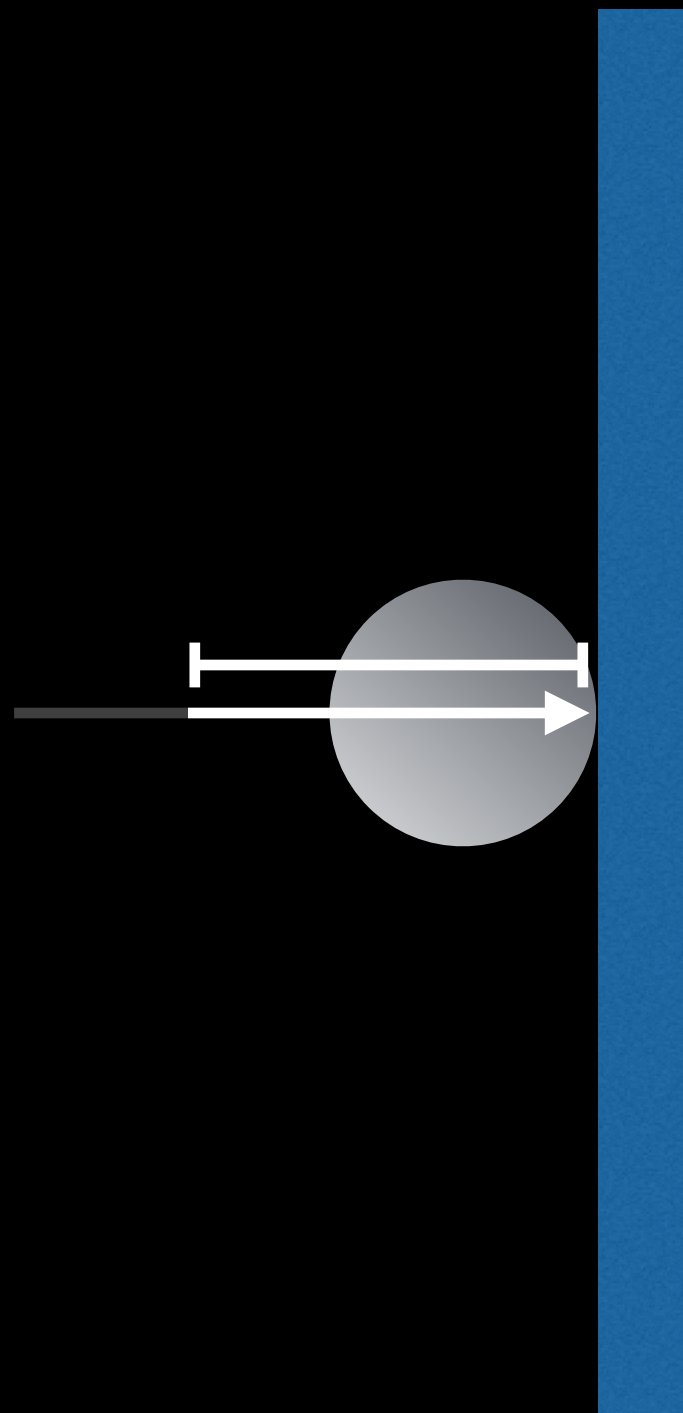




```
oldVel = body->vel
physicsStep()
if (impulse > limit)
    newBodies = fracture(body)
    for each body b in newBodies
        b->vel = oldVel*t + b->vel*(1-t)
    end
end
```

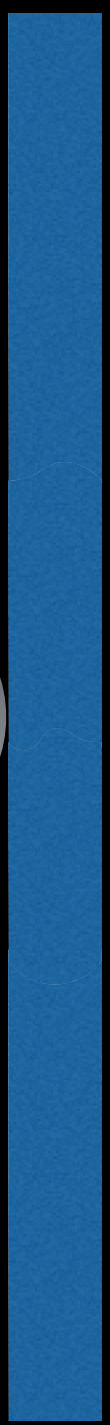
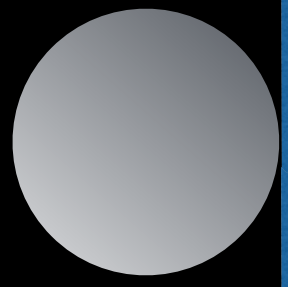
SMASH

HIT



SMASH

HIT





```
collisionDetection()
```

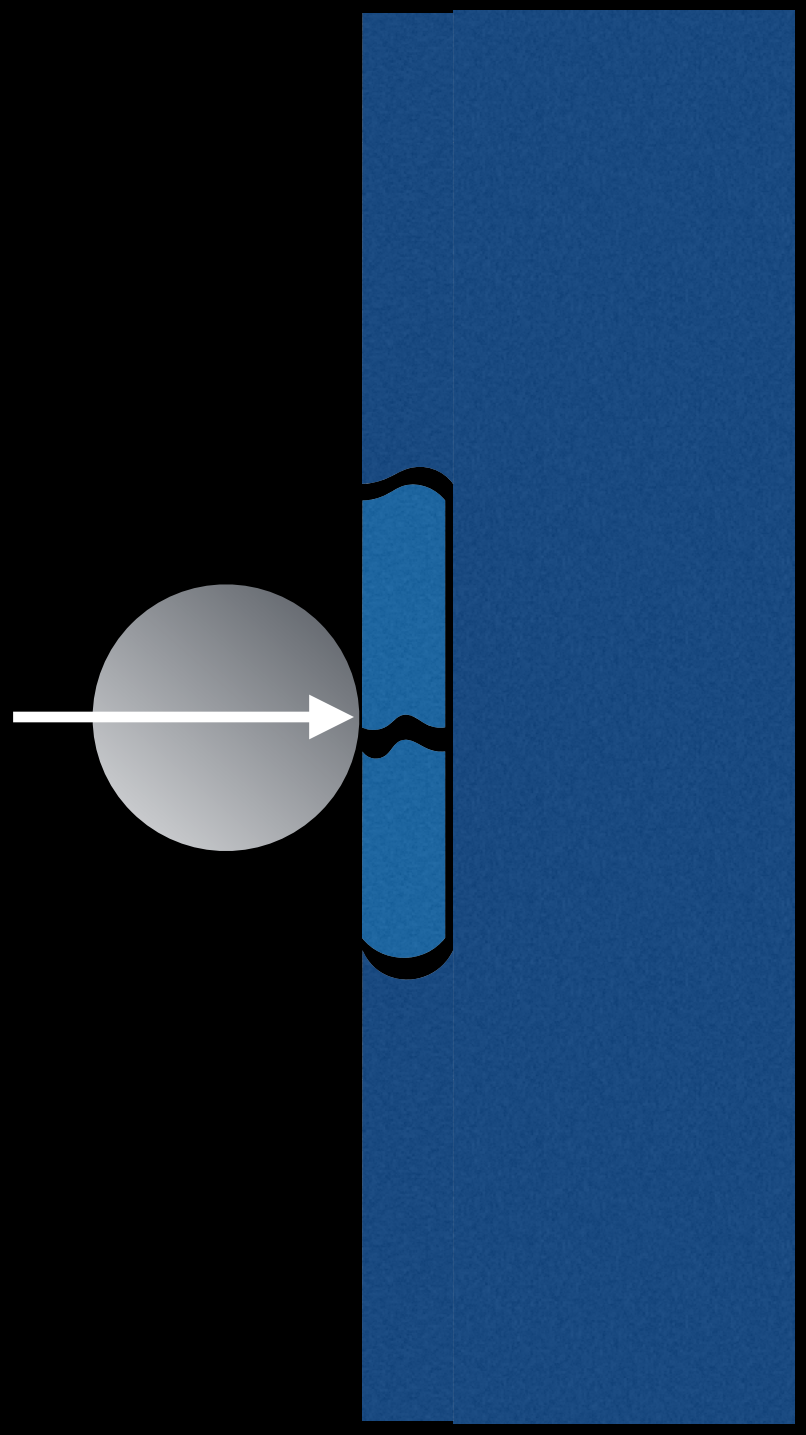
```
for each contact c
    if c involves fracture then
        c.maxImpulse = limit
    end
end
```

```
solver()
integration()
```

```
for each contact c
    if (c.impulse == limit)
        fracture(body)
    end
end
```

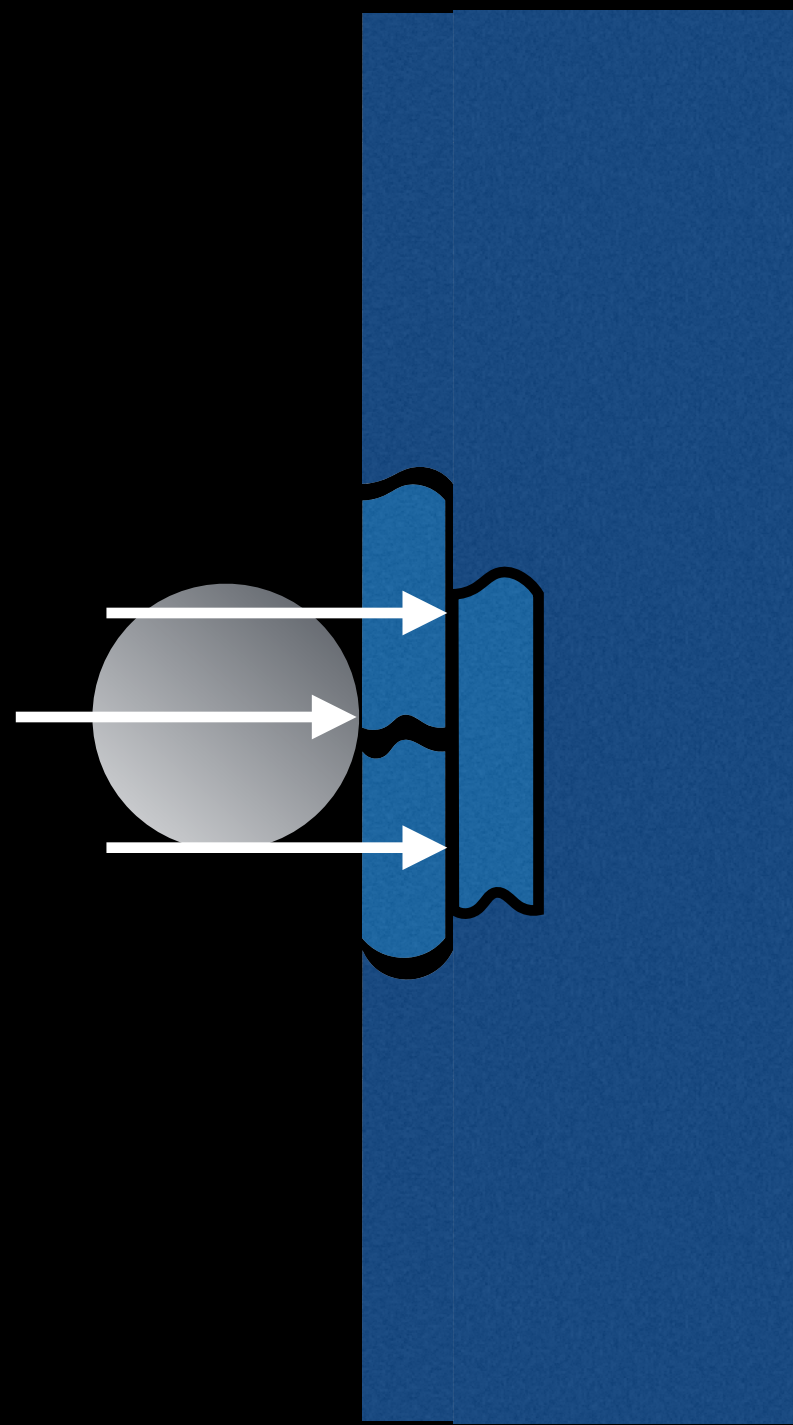
SMASH

HIT



SMASH

HIT





- 1 integrate new velocities
- 2 collision detection
- 3 limit impulses
- 4 run solver
- 5 if there are saturated impulses
 - fracture objects
 - collision detection on new objects
 - goto 3
- 6 integrate new positions



```
1 integrate new velocities
2 collision detection
3 limit impulses
4 run solver
5 if there are saturated impulses
    fracture objects
    collision detection on new objects
    if ++iterationCount < 3 then goto 3
6 integrate new positions
```

SMASH

HIT





Broad phase

Dynamic Bounding Volume Tree
World offset shifting



Near phase

GJK incremental manifold
Speculative contacts



Solver

Sequential Impulse
No solver islands
Custom deactivation



dennis@mediocre.se
tuxedolabs.blogspot.com